PIL Reference Spec

TABLE OF CONTENTS

APPENDICES

FIGURES

TABLES

1. INTRODUCTION TO PIL

1.1 What Is PIL?

PIL (Personal Interface Loop) is a system which permits communication from one device to another. In comparison with other interface systems PIL is small, low power, low cost, medium distance, and relatively slow. As the name implies devices are connected in a circular loop structure. Digital messages travel from one device to the next around the loop in one direction only. All devices must obey certain functional, electrical, and mechanical rules in order to communicate by means of PIL.

1.2 Some Application Examples

In order to help the new user understand the wide spectrum of capabilities and potential uses of PIL three possible applications are briefly described. These examples are fictitious and do not necessarily indicate that a particular product or system described either exists or is now or will be under development. Some of the products do exist, however, and the systems described here are readily acheivable given the software programs and devices. Hopefully the new user will gain some feeling of the range of problems which PIL can help to solve.

A university graduate student in forestry is working on his dissertation, a project studying the effect of various types of herbicides on underbrush, tree growth, etc. He makes observations and takes various data from several different plots miles from school. Because of its low cost and continuous memory feature he uses the HP 41C programmable calculator as a data collector. He has written a short program in the machine which prompts him for the correct data item and stores it along with all the other data from each area in separate registers in the HP 41C. The university forestry lab has purchased a small PIL system including the HP 82160A PIL module which plugs into the HP 41C, the HP 82161A minicassette tape drive, the HP 82162A 24 column thermal printer, and a plotter. When the student returns to the lab he plugs his calculator into the system and inserts a minicassette into the tape drive which contains his calculator programs and the data he has collected on his project over a period of several months. He loads and runs a program from the minicassette which reads the data taken today from the HP 41C and stores it on the tape with the rest of the data. He then loads another program which uses all the collected data on the tape to make finished graphic plots and histograms of various combinations of the data, such as toxicity levels over time, amount of undesirable foliage with respect to the type of herbicide, etc. Prior to the HP 41C and PIL, the cost of such an

application would have been prohibitive. Furthermore, PIL's low power requirements and loop power down feature permit unattended operation for extended time periods. If the student had needed to measure toxicity level, for example, every hour for a week, a small system in a weatherproof box could be left on site under battery power. Naturally, the measuring apparatus would need to be connected to the PIL system, but this would be facilitated with the HP 82165A PIL General Purpose Interface Device.

The second example deals with a small chain of three jewelry stores in nearby cities. Each store uses from two to five point-of-sale terminals each consisting of an HP 41C with a custom keyboard overlay and an HP 82162A 24 column thermal printer. These devices are all connected via PIL to two HP 82161A minicassette tape drives and an 80 column printer located in the back room of each store. Each of the calculators contains a program which prompts the salesclerk for the necessary information for each transaction. The data, which might include item description, inventory number, price, charge card number, etc. is stored on one of the minicassette drives as each calculator takes its turn controlling the PIL system. A customer receipt is printed on the small printer on the counter with the calculator and a transaction summary is printed on the larger printer simultaneously in the back room. At the end of the day's business the manager can place a master inventory tape in the second tape drive and run a special program to update it based on the transactions of the day in the other drive. Other special programs might generate sales reports, a list of low inventory items to be re-ordered, etc. With the HP 82164A PIL RS232C Interface Device and a modem the transaction or inventory data could even be transmitted over the telephone to the main store. Once again, prior to PIL such a system would have been much too expensive for most very small businesses.

A small electronics firm is about to begin volume production of a new audio amplifier printed circuit board. The production engineer has designed an automatic test system consisting of the HP 85A personal computer with the HP 82901A minifloppy disc drive, the HP 82938A PIL Interface Module, and the HP 3468A PIL Digital Voltmeter. The engineer has also built a special device containing a test fixture for the PC boards, a programmable power supply, a programmable waveform generator, and relay switches to connect these devices to the various PC board test points. This special device uses the HP 82166A PIL General Purpose Interface Module so that it too is controlled via PIL. After initial debugging the engineer writes the test program so that up to three identical test stations can be controlled on the same loop since he has found that PIL and the HP 85A are fast enough to support this throughput. Test programs and results are stored on the minifloppy drive for future failure analysis. Later when a second version of the amplifier board goes into production some simple program modifications permit testing of both versions at the same time on different test stations on the same system. The low cost and flexibility of PIL were the deciding factors in the use of automated testing as opposed to traditional manual testing.

1.3 Functional Overview

PIL is a master-slave interface system. One of the devices on the loop is designated the loop controller and this device has the responsibility to transmit all commands to other devices on the loop. The HP 41C and the HP 85A are examples of devices that can be PIL controllers when they are equiped with the proper plug-in module. A device with the ability to send data rather than commands to other devices on the loop is called a talker device. Note that even though a device has the talker capability it must not actually send its data until commanded to do so by the loop controller. The HP 82161A minicassette tape drive is a device which can be a talker. Listeners are devices with the capability to receive data from the loop. Once again, listeners must remain inactive until they receive a command from the controller which enables them to receive data. The HP 82162A Printer is an example of a device which can receive data.

Controller, talker, and listener are the three basic capabilities of PIL devices. A device may have only one of the three or may include some combination of capabilities as is more often the case. Talkers often have listener capability and controllers almost always have both talker and listener attributes as well. In general, every PIL system has all three capabilities somewhere within its devices, however, under special circumstances a system can be constructed with only a talker and one or more listeners without the controller. In this case, all devices must be able to operate in the talk-only or listen-only mode. An example might be a voltmeter logging readings on a printer. In larger loop systems there are usually several talkers and listeners. The controller will only permit one of the talkers at a time to be active, but may enable multiple listeners to receive the talker's data if desired. There may even be several controllers. In this case one of the controllers (the system controller) is in charge when the system is first turned on. Protocol exists within PIL to allow the various controllers to take turns controlling the other devices on the loop. Only one of the controllers at a time is active.

In addition to these three basic capabilities, PIL provides a device with others as well. There are two different ways for a device to indicate to the controller that it needs service. The power down function permits devices to go into a low power mode and to return to full power operation on command from the controller. There are three different ways for the controller to automatically assign addresses to the various devices on the loop. Devices can implement one byte addresses; this permits as many as 31 devices to share the loop. If need be, a two byte address can be used which provides the possibility of as many as 961 devices. Devices can tell the controller what type of device they are and what capabilities they have. Instruments can be triggered or cleared and can be commanded to respond to their panel controls or to programming messages over the loop. Controllers can interrupt long transmissions between a talker and listener temporarily to perform other tasks. This wide range of capabilities permits the controller to be in charge of not only the operation of the loop, but also its configuration. It is possible to implement high-level functions in the controller

device which make the details of loop protocol entirely transparent to the user. For example, a calculator, a printer, and several other devices could comprise such a "friendly" system in which the user simply executes a "print" function. The calculator automatically configures the loop, finds the printer, and sends the commands which perform the appropriate task.

The discussion so far has indicated two major classes of messages which are sent over the loop. Interface messages are usually sent by the active controller and are used to control the operation of the system. The controller's commands are an example of this type of message. In general, these messages are considered overhead. The primary reason for having PIL is the transmission of the second type of message. These are called device dependent messages and they do not directly affect the operation of the interface system though they use it as a communication medium. The data transmitted from the talker to the listener exemplifies this type of message.

Normally only one message is in transit around the loop at any given time whether it be an interface message or a device dependent message. In general, when a device sources a message it waits for the message to go completely around the loop and return before transmitting the next message. This is part of the process called the loop handshake and it guarantees that talkers and controllers do not send messages faster than the other devices can accept them. Each message must pass through each device. When a device receives a message, it does not pass it on to the next device until it is ready to receive the next message. Consequently, when the message returns to the sourcing device, it knows for certain that any and all other devices have received this message and are ready for the next. This also provides an excellent error-checking capability. If the message returns the same as it was sent, the sending device knows that all devices must have received it correctly.

Clearly, if there are a number of slow devices on the loop and they each delay the message for a little while until they are ready and then pass it on to the next device which does the same, the loop speed will be unacceptably low. Fortunately, data messages are intended for only one or possibly two devices at a time. Other devices on the loop are in their inactive state. Devices are built so that messages not meant for them are passed on very rapidly. In this manner the loop throughput can be maintained at an acceptable level.

However, commands are often meant for all devices on the loop at the same time, so in order to maintain reasonable speed the handshake process is modified slightly. When a device receives a command message it passes it on immmediately but also retains a copy of the message and begins to execute or interpret the command also. In this manner all devices can be executing the same command at about the same time. The message returns to the controller fairly rapidly but in this situation its return does not indicate that the devices are ready. The controller must now send a special message called "ready for command". This message is held up by each device until it is ready. If all devices execute the command in about the same time, there will be

a delay until the first device is finished and then the "ready for command" message will move around the loop fairly rapidly and return to the controller. Now the controller knows that all devices are ready and the next message can be sent. This modified handshake helps keep up the loop speed even when commands are for all devices simultaneously.

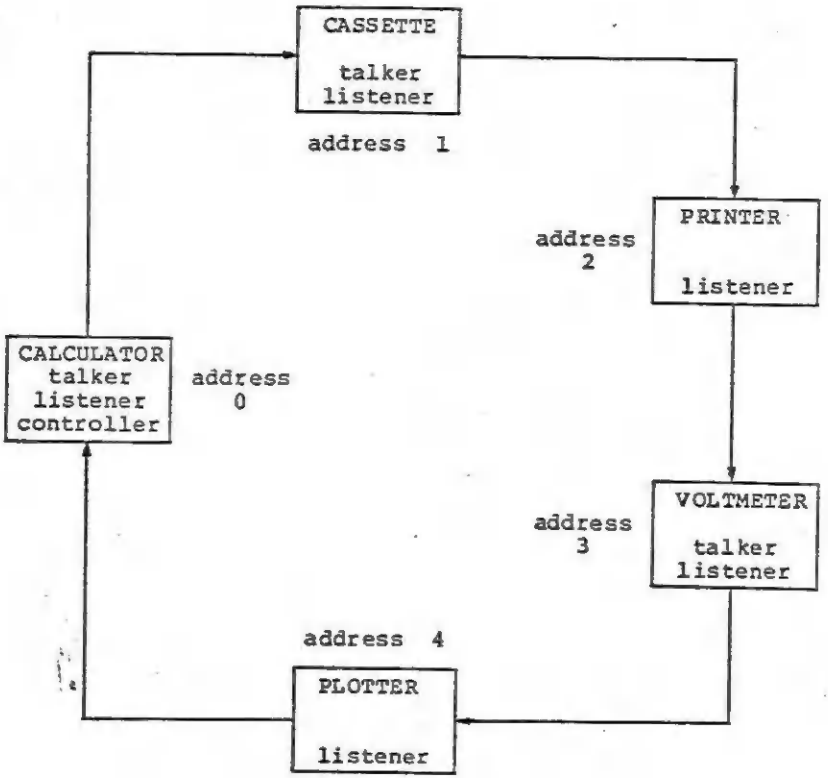


FIGURE 1-1: PIL SYSTEM EXAMPLE

Suppose that we have a PIL system as shown in figure 1-1. The devices with their basic capabilities, loop addresses, and direction of message travel are shown. Suppose that the user wants to print a voltmeter reading on the printer. For simplicity, assume that the devices have previously been assigned addresses as shown and that the voltmeter has already taken the reading and is merely waiting to transmit it to the printer. The

calculator first sends an "unlisten" command which causes any previous active listener devices to go inactive. When the command returns the calculator sends the "ready for command" message as is required after every command. The calculator knows that the voltmeter has been assigned address 3 and it now sends a command which causes device number 3 to be the talker. Other devices recognize that the command does not contain their address and retransmit it without taking any action. Even though the voltmeter is now addressed to talk it does not yet send its data. This command also causes any previously addressed talker to go inactive since the loop may only have one talker at a time. When the talk address command and the following "ready for command" have returned to the calculator, it sends a "listen address number 2" command. The printer alone recognizes this command in similar manner and after the "ready for command" may receive data messages. The only difference is that this command does not cause other listeners to go inactive, hence the need for the "unlisten" command at the beginning. The calculator now sends a special message called "send data" which causes the addressed talker to actually begin sourcing data messages. The "send data" is in a class of messages called ready messages which are used for handshake purposes. When the voltmeter receives the "send data" it does not retransmit it but instead sends its first data message. The voltmeter sends a sequence of message bytes, for example +2.658VDC followed by a carriage return, line feed pair. Each data byte goes around the loop to the printer, which holds the byte momentarily until it is loaded into the buffer, and back to the voltmeter. After each byte is received and error-checked the voltmeter sends the next character. The carriage return and line feed cause the printer to actually print the contents of its buffer. When the line feed returns and error-checks properly the voltmeter sources a special "end of transmission" ready message. This message signals the controller that the transmission is now complete and that the controller should resume active control of the loop. Similar to the "send data" message, the calculator does not retransmit the "end of transmission" but replaces it with the next command message.

It must be noted that PIL protocol is based very strongly on HP-IB (Hewlett-Packard's implementation of IEEE-488). While it appears that PIL is functionally a bit-serial version of HP-IB, the user should be warned that there are some significant differences in protocol. While familiarity with HP-IB is very helpful, it is not necessary. All information needed to learn and use PIL is contained in this document.

1.4 Electrical Overview

Every message on the loop is sent as a sequence of eleven bits called a message frame. The very first bit, called the sync bit, is coded in a special way so that each device can easily recognize the beginning of a frame. The sync bit and the next two bits sent are called control bits and they determine the major classification of the frame. Command messages, ready messages, and data messages are examples of these major classes. The remaining eight bits, sometimes called data bits (not to be

confused with a data message frame) specify the particular message within the classification. The "unlisten" command frame, for example, would be 100 0011111, while the "interface clear" command is 100 10010000. The "send data" ready frame is 101 01100000. The space between the control and data bits is for clarity only and does not represent a time delay or any other kind of delimiter.

The electrical connection from one device to the next is a differential, voltage-mode, two-wire, balanced line. Both wires are floating with respect to both devices' ground connections. One of the wires is chosen as a reference and the voltage of the other wire is measured only with respect to the reference. This has several advantages. Device grounds need not all be at the same potential. In fact there can be rather large differences with no effect. This is especially handy for voltmeters which need to measure values not referenced to ground. It totally avoids the problem of ground loops in an interface system. Most noise pulses tend to affect both wires equally; this type of common mode signal is very strongly rejected in a balanced system like PIL. The same holds true for noise radiation from the system itself. As one wire's voltage rises, the other's falls, tending to cancel any radiated signal.

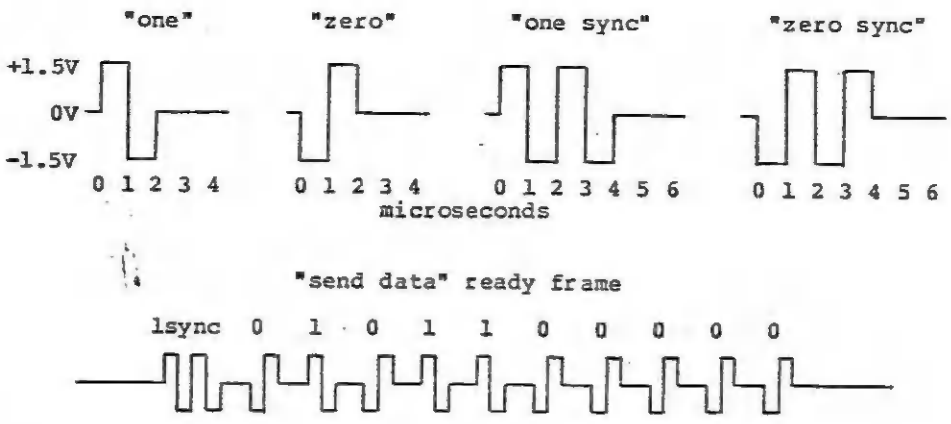


FIGURE 1-2: PIL BIT ENCODING

Bits are encoded using a three level, or bipolar, code. The voltage difference between the two wires may be nominally -1.5 Volts, 0 Volts, or +1.5 Volts. A logical one is encoded as a high pulse (+1.5 Volts) followed by a low pulse (-1.5 Volts). A logical zero is a low followed by a high. A logical one sync is a high, low, high, low sequence. A zero sync is low, high, low,

high. The nominal pulse width is one microsecond and each bit sequence is always followed by a minimum delay time (0 Volts) of about two microseconds. Refer to figure 1-2. This type of code provides good noise immunity, is relatively insensitive to speed and speed changes, is self-clocking, and has no DC component in the signal.

Frames are asynchronous with respect to each other. That means no common system clock is necessary in PIL. Furthermore, even bits within frames are asynchronous. One can see that sync bits require six microseconds to transmit while other bits take four microseconds. A complete frame, which may contain one byte of data, takes 46 microseconds assuming no extra delay between bits. If we further assume no extra delay between frames, ■ maximum loop rate of slightly over 20 kilobytes per second could be achieved. In ■ typical implementation with present electronics, other hardware and software delays lower this rate to somewhere between three and five kilobytes per second. This is enough to transmit more than ■ full single-spaced typewritten page of data every second.

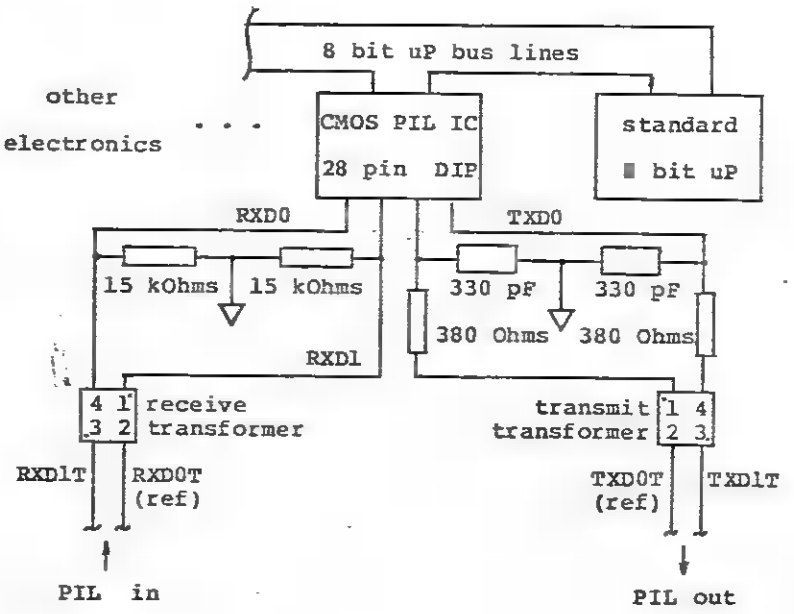


FIGURE 1-3: EXAMPLE DEVICE ELECTRONICS

Figure 1-3 is a simple block diagram of the interface electronics used in a particular implementation of PIL in ■ device. The PIL input and output lines are isolated through

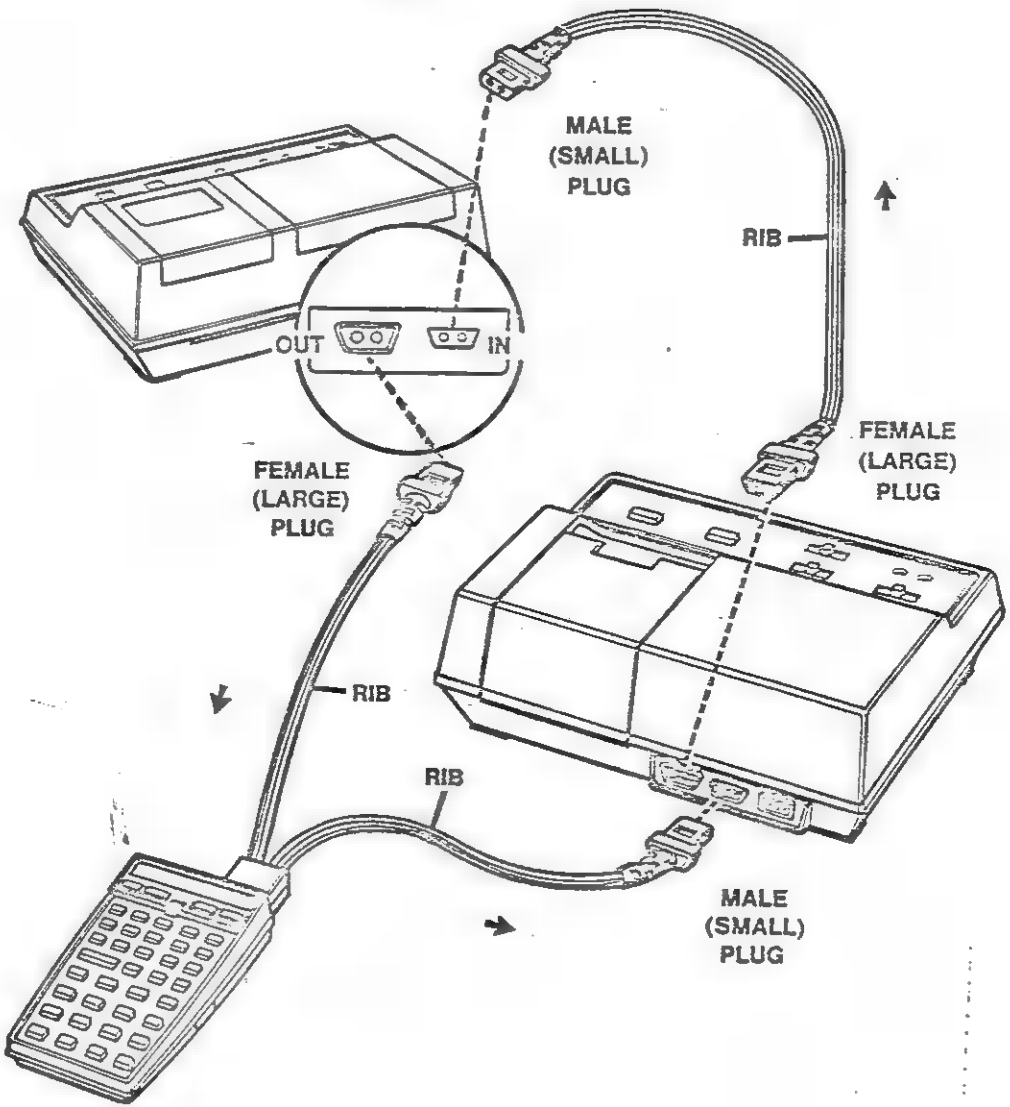


FIGURE 1-4: PIL CONNECTORS

simple pulse transformers. The discrete components provide the proper load value for input and filtering for EMI reduction with impedance matching for the output side. Most of the PIL protocol and interface functions are implemented in the single CMOS LSI PIL integrated circuit in a 28 pin DIP package. The remaining parts of the interface functions are implemented in firmware using any one of several standard microprocessors or one-chip microcomputers. The microprocessor is used to control the device functions and the interaction between the device and the interface chip as well. Since most devices will require the microprocessor anyway, the cost of this chip or chips is really only partly attributable to the interface and partly to the device itself.

1.5 Mechanical Overview

Since PIL utilizes a loop structure, each device only needs two connectors, an input and an output, regardless of how many devices are on the loop. In the interest of very small size, PIL uses a special connector. Both connectors panel mounted together only require about 0.4 in. by 1.5 in. of panel space. The unit is about 0.8 in. deep. If less space than this is available the lines themselves can go directly into the panel through strain reliefs with cable connectors at the loose ends. connectors are constructed so that the reference line cannot be reversed and also so that an output cannot be plugged into another output. Cable connectors plug together properly to form running cable "splices". See figure 1-4.

For relatively short distances up to a maximum of 10 meters from one device to the next, very low cost "zip cord" can be used as shown in the illustration. Note that the "rib" on the "zip cord" always corresponds to the reference line. For longer distances up to 100 meters between devices, shielded twisted pair cable must be used. No change is necessary in the device electronics for the longer distance. Only the cable between those particular devices that are farther than 10 meters apart must be changed.

2. FUNCTIONAL SPECIFICATIONS

2.1 Functional Partition

A PIL device may be partitioned conceptually into three functional areas each with unique capabilities: device functions, interface functions, and message coding logic. Refer to figure 2-1.

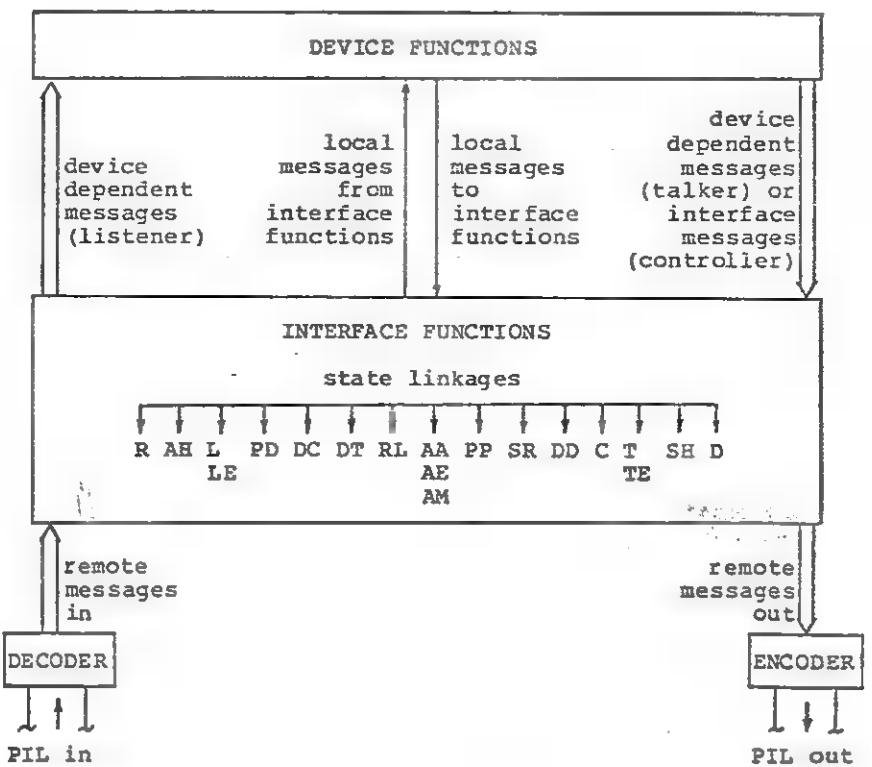


FIGURE 2-1: FUNCTIONAL BLOCK DIAGRAM

Device functions are those unique features and capabilities that each device possesses. These are entirely up to the device

designer. The device functions communicate with the interface functions via local messages. The interface functions are not designer-specified, but are defined in this chapter. The interface functions within a physical device must conform to the specifications exactly in order for the device to properly communicate with other devices via the PIL system. The message decoding and encoding must also be the same as described here for the various devices to exchange data unambiguously.

Through the interface functions the device may receive, process, and send messages. The interface function is the basic operational unit and each individual interface function may receive, process, and send only a limited set of messages. Each of the interface functions is defined in terms of one or more groups of interconnected, mutually exclusive states, called a state diagram. Within a single group of interconnected states one and only one state may be active at any given time. The logical variable associated with a state has a true value if and only if that state is active. Definitions are given for each state of each interface function describing the messages that must be sent while that state is active and the conditions under which a transition must occur from that state to another state within the group. The total capability of the device depends on the collection of allowable subsets of the interface functions which the designer decides to implement. The state diagrams define the external behavior of the interface functions and are not intended to limit the design. Interface functions and combinations of interface functions may be implemented with any set of hardware, firmware, and/or software which the designer chooses, provided the external behavior is as specified. Table 2-1 lists the interface functions and their mnemonics.

TABLE 2-1: INTERFACE FUNCTIONS

Interface Function	Mnemonic
receiver	R
acceptor handshake	AH
listener or extended listener	L or LE
power down	PD
device clear	DC
device trigger	DT
remote local	RL
automatic address	AA
automatic extended address	AE
automatic multiple address	AM
parallel poll	PP
service request	SR
device dependent commands	DD
controller	C
talker or extended talker	T or TE
source handshake	SH
driver	■

Each message represents a quantity of information and the logical value of each message is received either true or false at any specific time. All interaction between an interface function and its environment is via messages sent or received. Messages sent between interface functions and device functions are called local messages. Local messages sent by device functions must exist for enough time to cause the required state transitions. The set of local messages sent from the device functions to the interface functions is defined and may not be changed by the designer. Local messages sent from interface functions to device functions, however, are completely designer specified. The logical value of any state or combination of states may be used to derive a local message sent to the device functions. Remote messages are those sent via the interface system. Encoding and decoding is the process of translating remote messages to and from interface signal line values. Interface functions must ignore any message coding not specifically defined. Pseudomessages (not shown in figure 2-1) are a small set of messages similar to local messages which are sent by the message coding logic to the interface functions. There is extensive interaction among the various interface functions, and this is accomplished by means of state linkages from other interface functions. A state linkage is a logical interconnection of two interface functions where the transition to an active state of one interface function is dependent on the existence of a specified active state in another interface function. In summary, an interface function's inputs are remote messages, pseudomessages, state linkages, and local messages from device functions; its outputs are remote messages and local messages as defined by the designer. A good understanding of these concepts together with figure 2-1 is essential to understanding the state diagrams and the entire interface system.

As discussed previously, remote messages (prior to decoding) consist of message frames of eleven bits; three control bits which place the message in one of several major categories, and eight data bits which specify the particular message within the category. These bits are sent in the following order and designated as shown:

C3 C2 C1 D8 D7 D6 D5 D4 D3 D2 D1

Note that C3 is also the sync bit indicating the start of a frame. There are four major categories of messages. The remaining combinations of control bits are used for other purposes. The categories are:

1. Data or end messages (DOE). These are the device dependent messages discussed earlier which are normally sourced by the active talker and received by the active listener(s). End messages use one of the control bits (C2) to indicate an end-of-record condition (not end of transmission). End messages are otherwise the same as any other data message.

2. Command messages (CMD). These interface messages are always sourced by the active controller. Sub-classes of commands as defined by the coding of the

data bits are: universal commands to which all devices respond, addressed commands to which only devices addressed to talk or to listen respond, talk and listen addresses (all devices), and secondary addresses (devices enabled by a primary address).

3. Ready messages (RDY). These interface messages are sometimes sourced by the active controller and sometimes by the active talker. They are used for handshake purposes, for example, to initiate or terminate a data transmission. Sub-classes of ready messages are: the ready for command message, the addressed ready group which includes start and end of transmission messages, and the auto address group.

4. Identify messages (IDY). These messages are normally sent by the controller to determine if devices have a need for service. Note that bit C1 of both DOE and IDY messages may be used by devices to indicate a need for service.

Complete information regarding coding of remote messages is contained in the last section of this chapter as well as in the appendices.

2.2 Notation

Each state that an interface function can assume is represented graphically as a circle. A four-letter upper-case mnemonic always ending in S is used within the circle to identify the state. All permissible transitions between states of an interface function are represented graphically by arrows between them. Each transition is qualified by an expression whose value is either true or false. The interface function shall remain in its current state if all expressions which qualify transitions leading to other states are false. The interface function shall enter the state pointed to if and only if one of these expressions becomes true.

An expression consists of one or more local messages, pseudomessages, remote messages, or state linkages used in conjunction with the operators AND, OR, or NOT. A local message to an interface function is represented by a lower-case three-letter mnemonic, for example, rdy. A pseudomessage is represented by a lower-case four-letter mnemonic, for example, sync. A remote message is represented by a three-letter mnemonic in upper-case, for example, IFC. A linkage from another state diagram is represented by the state mnemonic enclosed in braces, for example, {LACS}. A state linkage is true if the enclosed state is currently active; otherwise, it is false. The AND operator is represented by a dot (.). The OR operator is represented by a plus sign (+). The NOT operator is represented by a horizontal bar placed over the portion of the expression to be negated. The resulting negated expression has a true value if and only if the value of the expression under the bar is false. The AND operator takes precedence over the OR operator in an

expression unless indicated otherwise by parentheses.

If a portion of an expression is optional in that its true value is not required for the complete expression to be true (at the designer's choice), then it is enclosed within square brackets [...]. If a specific expression causes a transition to ■ state from all other possible states of the diagram, a shorthand notation is used instead of all the individual transitions being drawn. An arrow without a state at its origin is used to represent this condition, and is assumed to originate in all states. Note also that the power-off state is a valid state of most interface functions and should normally be shown with a transition leading to the state to be entered when power is first turned on. This state is omitted for clarity in the diagram and only the transition to the first state is shown.

2.3 R (Receiver) Interface Function

General Description

The R interface function provides the device with the capability to receive messages over the interface. It categorizes the received message into major classifications which are used by the AH (Acceptor Handshake) and D (Driver) interface functions to effect proper communication.

REIS (Receiver Idle State)

In REIS the R interface function is not engaged in any message transfer process. The function powers on in this state. Transition out of this state does not occur until a valid sync bit has been received from the loop.

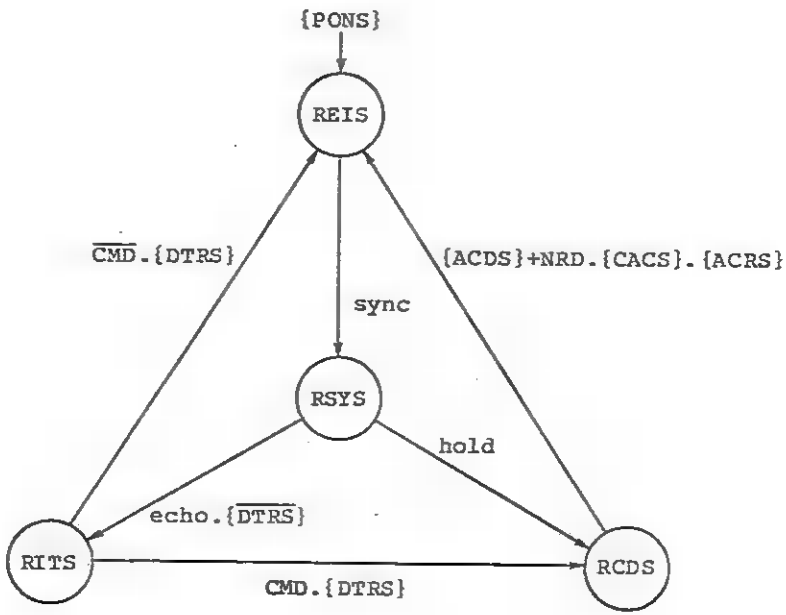
RSYS (Receiver Sync State)

In this state the R interface function has begun to receive a message from the interface and is in the process of determining whether to immediately retransmit the message (echo) or not (hold).

RITS (Receiver Immediate Transfer State)

DOE messages not sourced by or intended for this device, CMD or IDY messages not sourced by this device, and ARG messages not for this device should be immediately retransmitted on the loop. In RITS the R function is signaling the D (Driver) function that retransmission of the incoming frame should commence. RITS is not entered until the D function is finished transmitting any previous message and is not exited until the D function has accepted the current message and commenced retransmission. Most messages which are echoed do not affect the device or interface functions (except R and D), however CMD messages are both echoed and held for device and interface interpretation. This decoding is also done in RITS. If the message is not CMD the function returns to REIS, otherwise the transition is to RCDS.

FIGURE 2-2: R STATE DIAGRAM



echo = DOE.{$\overline{TACS}$}.{$\overline{SPAS}$}.{$\overline{DIAS}$}.{$\overline{AIAS}$}.{$\overline{LACS}$}.{$\overline{CACS}$}
+(CMD+IDY).{$\overline{CACS}$}
+ARG.{$\overline{TADS}$}.{$\overline{TACS}$}.{$\overline{SPAS}$}.{$\overline{DIAS}$}.{$\overline{AIAS}$}.{$\overline{LACS}$}.{$\overline{CACS}$}.{$\overline{CSBS}$}

hold = DOE.({TACS}+{SPAS}+{DIAS}+{AIAS}+{LACS}+{CACS})
+(CMD+IDY).{CACS}
+ARG.({TADS}+{TACS}+{SPAS}+{DIAS}+{AIAS}+{LACS}+{CACS}+{CSBS})
+RDY.$\overline{ARG}$

TABLE 2-2: R MNEMONICS

Messages

sync - valid sync bit received

ARG - addressed ready group
CMD - command
DOE - data or end
IDY - identify
NRD - not ready for data
RDY - ready

Interface States

RCDS - receiver data state (links to AH, C functions)
REIS - receiver idle state
RITS - receiver immediate transfer state (links to D, C, PP functions)
RSYS - receiver sync state

AIAS - accessory identify active state (from T, TE function)
ACDS - acceptor data state (from AH function)
ACRS - acceptor ready state (from AH function)
CACS - controller active state (from C function)
CSBS - controller standby state (from C function)
DIAS - device identify active state (from T, TE function)
DTRS - driver transfer state (from D function)
LACS - listener active state (from L, LE function)
PONS - power on state (from PD function)
SPAS - serial poll active state (from T, TE function)
TACS - talker active state (from T, TE function)
TADS - talker addressed state (from T, TE function)

RCDS (Receiver Data State)

In this state the R function is signaling the AH (Acceptor Handshake) function that the message being received is for this device. When the AH function signals in return that it has accepted the message, the R function returns to REIS.

Remote Messages

This interface function is not capable of originating remote messages. It only receives messages and transfers them to other interface functions for appropriate action.

Additional Requirements and Guidelines

While it is possible to simplify the implementation of the R function by having it receive all eleven bits of the message frame prior to any decoding (echo, hold expressions), the resulting device will in general cause unacceptable loop

throughput. It is therefore strongly recommended that the R function do this decoding and execute the proper transitions as soon as enough bits have been received to permit the correct decision to be made. This occurs on the first bit for DOE messages, the third bit for CMD and IDY messages, etc.

For simplicity and clarity only three categories of messages which must be echoed are shown in the R function state diagram, although a few other cases could be included (for example, an auto address configure message in a device which is already configured). If desired these other cases may be included in the echo and hold expressions to increase loop throughput for these sequences. Great care should be taken if this is done to insure loop integrity.

Please note that all devices must implement the full R interface function capability. No subsets are permitted.

2.4 AH (Acceptor Handshake) Interface Function

General Description

The AH interface function receives messages from the R (Receiver) function which are for this device and indicates to the device and other interface functions when the message is valid for interpretation. The AH function also determines whether the message needs to be retransmitted after interpretation (repeat) or not (norepeat), and indicates to the D (Driver) interface function when this retransmission should begin.

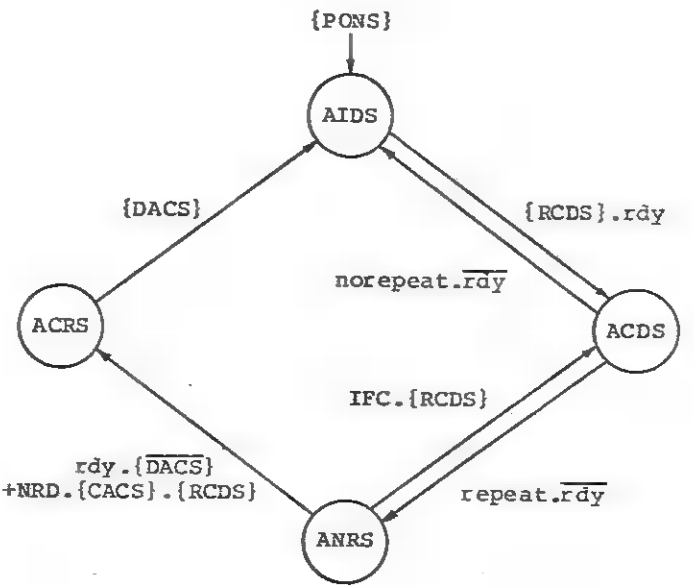
AIDS (Acceptor Idle State)

In AIDS the AH function is not engaged in the handshake cycle and does not have a message available. This is the power on state for this function. When the R (Receiver) function indicates that a message for this device is available, the AH function enters ACDS.

ACDS (Acceptor Data State)

In this state the interface function is indicating to all other interface and device functions that a message for this device has been received and is now valid for interpretation. This state also decodes the message to determine whether it should be retransmitted or not. DOE messages sourced by this device, CMD messages, RDY or IDY messages sourced by this device, and SOT messages for this device are not retransmitted on the loop. After the device has indicated that it has accepted the message for interpretation by sending the rdy local message false, the AH function returns to AIDS if retransmission is not necessary, or transitions to ANRS if the message must be retransmitted.

FIGURE 2-3: AH STATE DIAGRAM



norepeat = DOE.({TACS}+{SPAS}+{DIAS}+{AIAS})

+CMD

+(RDY+IDY).({CACS}+{CSBS})

+SOT.({TADS}+{TACS}+{SPAS}+{DIAS}+{AIAS})

repeat = DOE.({LACS}+{CACS})

+RDY.$\overline{SOT}$.$\overline{\{CACS\}}$.$\overline{\{CSBS\}}$

+SOT.$\overline{\{TADS\}}$.$\overline{\{TACS\}}$.$\overline{\{SPAS\}}$.$\overline{\{DIAS\}}$.$\overline{\{AIAS\}}$

TABLE 2-3: AH MNEMONICS

Messages

rdy - ready for next frame

CMD - command
DOE - data or end
IDY - identify
IFC - interface clear
NRD - not ready for data
RDY - ready
SOT - start of transmission

Interface States

ACDS - acceptor data state (links to R, SH, L, LE, T, TE, C, DC, DT, PP, AA, AE, AM, RL, PD functions)
ACRS - acceptor ready state (links to D function)
AIDS - acceptor idle state (links to PD function)
ANRS - acceptor not ready state

AIAS - accessory identify active state (from T, TE function)
CACS - controller active state (from C function)
CSBS - controller standby state (from C function)
DACS - driver transmit from acceptor state (from D function)
DIAS - device identify active state (from T, TE function)
LACS - listener active state (from L function)
PONS - power on state (from PD function)
RCDS - receiver data state (from R function)
SPAS - serial poll active state (from T, TE function)
TACS - talker active state (from T, TE function)
TADS - talker addressed state (from T, TE function)

ANRS (Acceptor Not Ready State)

In ANRS the AH function is waiting for the device to indicate (via rdy) that it is ready for the next message frame. This typically occurs after the device has completed any actions necessitated by the current message. When the device is ready the function transitions to ACRS. If the R function indicates that the IFC message has been received while the AH function is waiting in ANRS, the function transitions back to ACDS.

ACRS (Acceptor Ready State)

This state is indicating to the D (Driver) function that retransmission of the current message frame should begin. When the D function indicates in return that it has accepted the frame and has commenced transmission, the AH function returns to AIDS.

Remote Messages

The AH interface function is not capable of originating remote messages. It only accepts received messages from the R (Receiver) interface function and transfers them to the device and other interface functions for appropriate action.

Additional Requirements and Guidelines

The AH function may be implemented such that it receives all eleven bits of the message frame prior to any decoding (repeat, norepeat expressions). It is permitted, however, to perform this decoding earlier in the frame so that loop throughput may be considerably enhanced at the expense of greater complexity.

Please note that all devices must implement the full AH capability. No subsets are permitted.

2.5 SH (Source Handshake) Interface Function

General Description

The SH interface function gives the device the ability to properly control messages which originate within it primarily while the device is a talker or controller. It coordinates the message transfer from the device to the D (Driver) function.

SIDS (Source Idle State)

The SH function powers on in SIDS and in this state no handshake is taking place. The device is not sourcing frames. When the appropriate interface functions indicate that the device should begin to source messages (source) the function transitions to SGNS.

SGNS (Source Generate State)

In SGNS the device is in the process of generating the next message frame to be sourced. When the device indicates that the new message is available (nba local message) and the D (Driver) function indicates that it is not still transmitting a previously sourced frame, the function enters SDYS. If the sourcing device aborts the function will return to SIDS.

SDYS (Source Delay State)

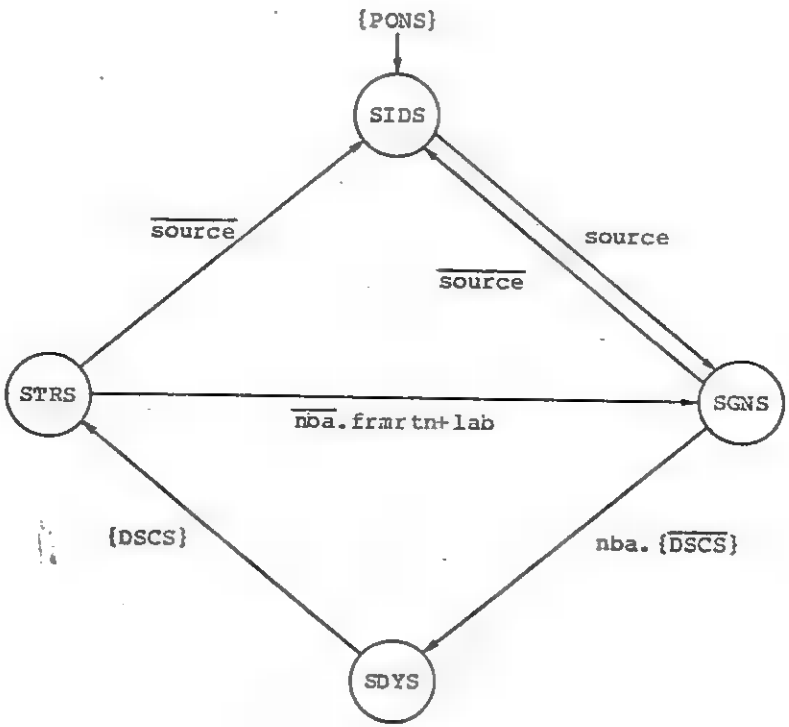
In this state the SH function is waiting for the D (Driver) interface function to acknowledge receipt of the sourced message and begin transmission. When this occurs the function enters STRS.

STRS (Source Transfer State)

In STRS the SH function is waiting for the sourced message to go around the loop and return (normal loop handshake, frmrtn expression). When this occurs and the device is generating a new

message frame (nba false) the function returns to SGNS. In addition, a controller may force the return to SGNS before the handshake is complete with the lab local message so that it may asynchronously source a new message frame. When the sourcing device has transmitted the last frame it will normally transition from STRS to SIDS since this frame is usually a RDY frame and does not use the same handshake cycle.

FIGURE 2-4: SH STATE DIAGRAM



source = {TACS}+{SPAS}+{DIAS}+{AIAS}+{CACS}+{ARSS}

frmrtn = (DOE.({TACS}+{SPAS}+{DIAS}+{AIAS})
+(CMD+RDY+IDY).{CACS}).{ACDS}

TABLE 2-4: SH MNEMONICS

Messages

lab - local abort	CMD - command
nba - new byte available	DOE - data or end
	IDY - identify
	RDY - ready

Interface States

SDYS - source delay state (links to D function)
SGNS - source generate state
SIDS - source idle state
STRS - source transfer state (links to T, TE, C, AA, AE, AM functions)

AIAS - accessory identify active state (from T, TE function)
ACDS - acceptor data state (from AH function)
ARSS - asynchronous request source state (from SR function)
CACS - controller active state (from C function)
DIAS - device identify active state (from T, TE function)
DSCS - driver transmit from source state (from D function)
PONS - power on state (from PD function)
SPAS - serial poll active state (from T, TE function)
TACS - talker active state (from T, TE function)

Remote Messages

The SH function does not originate remote messages on the loop but merely controls the orderly transfer of messages from the device to the D (Driver) function for transmission. The device may change the message frame while SIDS or SGNS are active, but must hold the frame valid while in SDYS and STRS.

Additional Requirements and Guidelines

Devices may be implemented which do not source messages; they are neither talkers nor controllers. The SH interface function is not present in these devices and this capability is designated SH0. Devices which do source frames must implement the full SH function as described; no subsets are allowed. This capability is designated SH1.

2.6 D (Driver) Interface Function

General Description

The D interface function provides the device with the

ability to transmit messages on the loop. These messages come from three different sources: messages generated by this device, messages received on the loop which must be immediately retransmitted, and messages received on the loop which must be retransmitted after acceptance by the device.

FIGURE 2-5: D STATE DIAGRAM

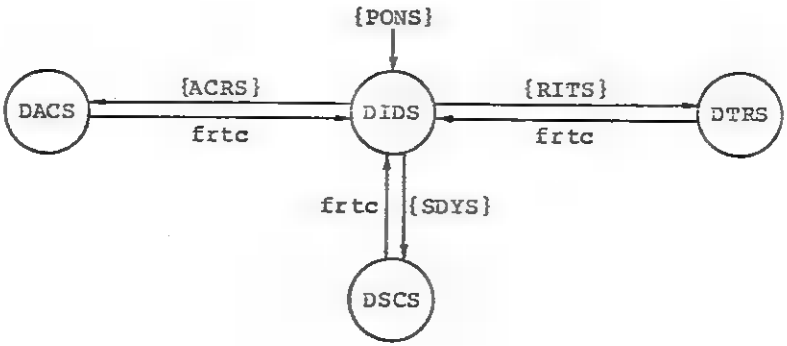


TABLE 2-5: D MNEMONICS

Messages

frtc - frame transmission complete

Interface States

DACS - driver transmit from acceptor state (links to AH function)
DIDS - driver idle state (links to PD function)
DSCS - driver transmit from source state (links to SH function)
DTRS - driver transfer state (links to R, PP functions)

ACRS - acceptor ready state (from AH function)
PONS - power on state (from PD function)
RITS - receiver immediate transfer state (from R function)
SDYS - source delay state (from SH function)

DIDS (Driver Idle State)

In DIDS the D function is not transmitting any message. The function powers on in this state.

DSCS (Driver Source State)

In this state the function is transmitting a message frame originated by this device. DSCS is entered when the SH (Source Handshake) function indicates that the message is ready to be transmitted. When transmission is complete the function returns to DIDS.

DTRS (Driver Transfer State)

In DTRS the D function has commenced retransmitting a message transferred from the R (Receiver) function which must not be held up by the device's ability to interpret it. When finished with the retransmission the D function returns to DIDS.

DACS (Driver Acceptor State)

This state is entered when retransmission of a message which has been received and interpreted by the device has begun. The message is transferred from the AH (Acceptor Handshake) function. When transmission is complete the function transitions to DIDS.

Remote Messages

The D function transmits all remote messages. These messages are transferred to it by the R (Receiver), AH (Acceptor Handshake), or SH (Source Handshake) function. In this transfer certain modifications of the message may take place. This is due to actions performed by the parallel poll, service request, and the various automatic address interface functions. Refer to these functions for a full description of this modification process.

Additional Requirements and Guidelines

All devices must have the full D function implemented. No subsets are permissible.

2.7 L (Listener), LE (Extended L) Interface Functions

General Description

These alternative interface functions provide the device with the capability to receive device dependent data including status, device ID, and accessory ID over the loop from other devices. This capability exists only when the device has been addressed to listen.

FIGURE 2-6: L STATE DIAGRAM

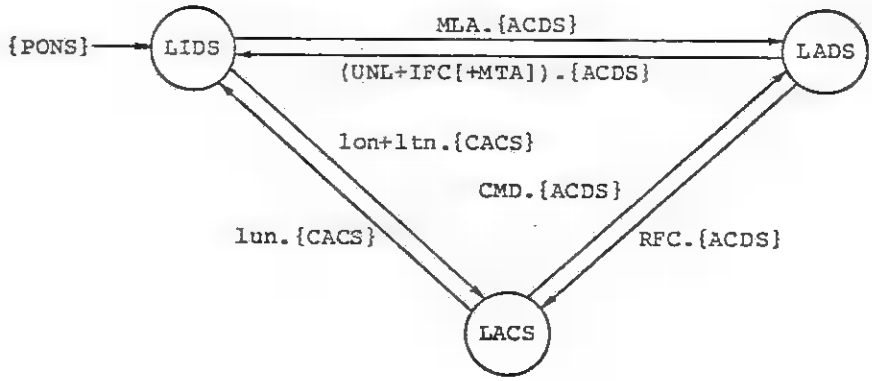


FIGURE 2-7: LE STATE DIAGRAM

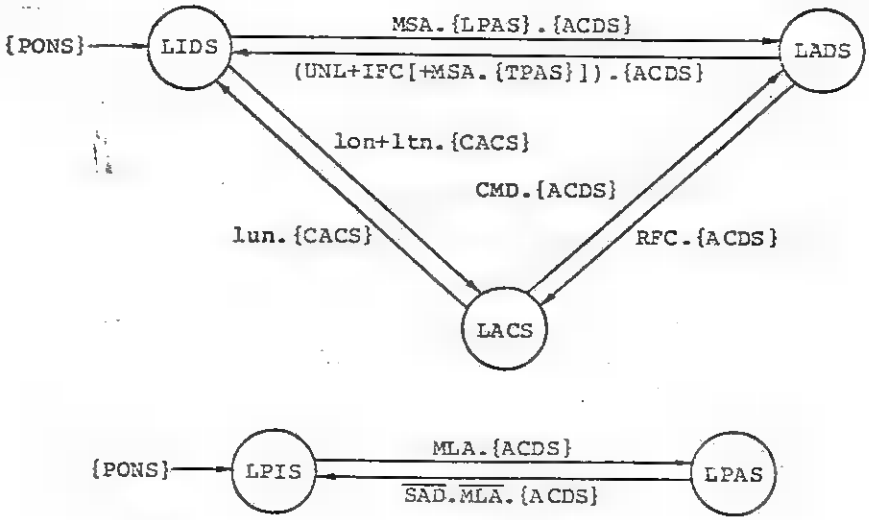


TABLE 2-6: L, LE MNEMONICS

Messages

lon - listen only
ltn - local listen
lun - local unlisten

CMD - command
IFC - interface clear
MLA - my listen address
MSA - my secondary address
MTA - my talk address
RFC - ready for command
SAD - secondary address
UNL - unlisten

Interface States

LACS - listener active state (links to R, AH functions)
LADS - listener addressed state (links to DC, DT, PP, RL, DD functions)
LIDS - listener idle state
LPAS - listener primary addressed state (links to LE, TE functions)
LPIS - listener primary idle state

ACDS - acceptor data state (from AH function)
CACS - controller active state (from C function)
LPAS - listener primary addressed state (from LE function)
PONS - power on state (from PD function)
TPAS - talker primary addressed state (from TE function)

There are two versions of this function. The L function uses one frame address and the LE function uses a two frame address. In all other respects these two alternatives provide the same capability. Due to the extensive similarity between these two variations both are described concurrently throughout this section. Only one of the two alternatives need be implemented in any specific device.

LIDS (Listener Idle State)

In this state the device is not able to receive device dependent messages. The function powers on in this state. A listen-only device or a controller device may use the appropriate local message (lon, ltn respectively) to enter LACS directly from LIDS without first being addressed.

LADS (Listener Addressed State)

In LADS the L function has received its listen address or the LE function has received both its primary listen address and secondary address over the loop. The function is prepared for, but not yet engaged in the receipt of device dependent messages

over the loop. On receipt of the RFC message following the listen address command the function enters LACS.

LACS (Listener Active State)

The device is enabled to receive device dependent messages over the loop while in LACS. This message transfer is controlled by the AH (Acceptor Handshake) interface function. If the function receives a CMD message while in this state it returns to LADS. When the ensuing RFC is received it returns to LACS unless the CMD is one which causes the function to become unaddressed. In this case the transition continues back to LIDS. A controller device may use the lun local message to abort the function from LACS to LIDS directly.

LPIS (Listener Primary Idle State)

In LPIS the device is able to recognize its primary listen address but is not able to respond to its secondary address. The LE function powers on in this state.

LPAS (Listener Primary Active State)

The function has received its primary listen address and is now able to respond to and recognize its secondary address in this state. The receipt of any command other than MSA or MLA (again) will cause return to LPIS.

Remote Messages

The L, LE functions do not have the capability to originate remote messages. While in LACS this function permits the device to receive and interpret device dependent messages (DOE) as controlled by the AH (Accep Handshake) interface function. While LACS is not active the device may not receive device dependent messages.

Additional Requirements and Guidelines

Since a fixed address may make it difficult or impossible to use a device in a particular loop configuration, it is strongly recommended that a means be provided for the user to change the device's address (and secondary address, if applicable). This could take the form of address switches or the inclusion of one of the automatic address interface functions, for example.

To enhance the compatibility between the various devices on the loop it is strongly recommended that wherever possible device dependent messages be sent using the ASCII code.

The interruption of a device receiving data by transitions in and out of LACS should not adversely affect the future receipt of input data. It is recommended that upon return to LACS the device should continue the input data string at the point of interruption. This is only true for a data string; device ID, accessory ID, and status strings must always restart at the beginning when interrupted.

Please note that a device's listen address and talk address may or may not be the same value.

Devices which do not implement the L or LE function will be designated L0 or LE0 respectively. The basic L or LE capability is designated L1 or LE1 respectively. If the L or LE function includes the ability to operate in the listen-only mode, the number 2 is added to the designation. Devices which have this capability must provide a switch to generate the lon local message. Devices which do not have the listen-only mode must always send the lon local message false. If the L or LE function has the ability to become unaddressed if the device's talk address is received, the number 3 is added to the designation. Devices with this capability must include the optional term ([+MTA] for L, [+MSA.{TPAS}] for LE) in the state diagram implementation; devices without this capability must not. Note that this capability requires the presence of the T or TE function. For example, a basic listener with listen-only mode would be written L1,2. An extended listener with unaddress capability would be LE1,3. A basic listener with full capability would be designated L1,2,3.

2.8 T (Talker), TE (Extended T) Interface Functions

General Description

These alternative interface functions provide the device with the capability to send device dependent data including status, device ID, and accessory ID over the loop to other devices. This capability exists only when the device has been addressed to talk.

There are two versions of this function. The T function uses a one frame address and the TE function uses a two frame address. In all other respects these two alternatives provide the same capability. Due to the extensive similarity between these two variations both are described concurrently throughout this section. Only one of the two alternatives need be implemented in any specific device.

TIDS (Talker Idle State)

In this state the device is not able to send device dependent messages. The function powers on in this state. A talk-only device or a controller device may use the appropriate local message (ton, tlk respectively) to enter TACS directly from TIDS without first being addressed.

TADS (Talker Addressed State)

In TADS the T function has received its talk address or the TE function has received both its primary talk address and secondary address over the loop. The function is prepared for, but not yet engaged in the sending of device dependent messages over the loop. When the appropriate SOT (start of transmission) message is received the function enters one of the active states

FIGURE 2-8: T STATE DIAGRAM

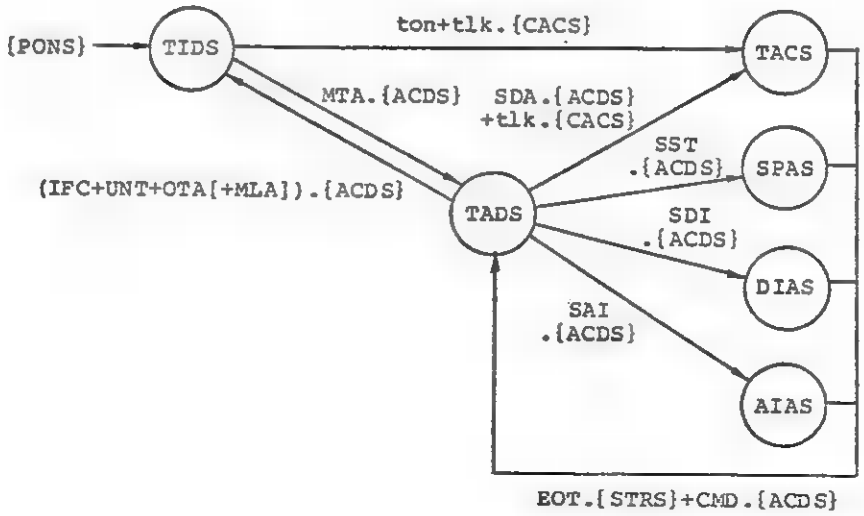


FIGURE 2-9: TE STATE DIAGRAM

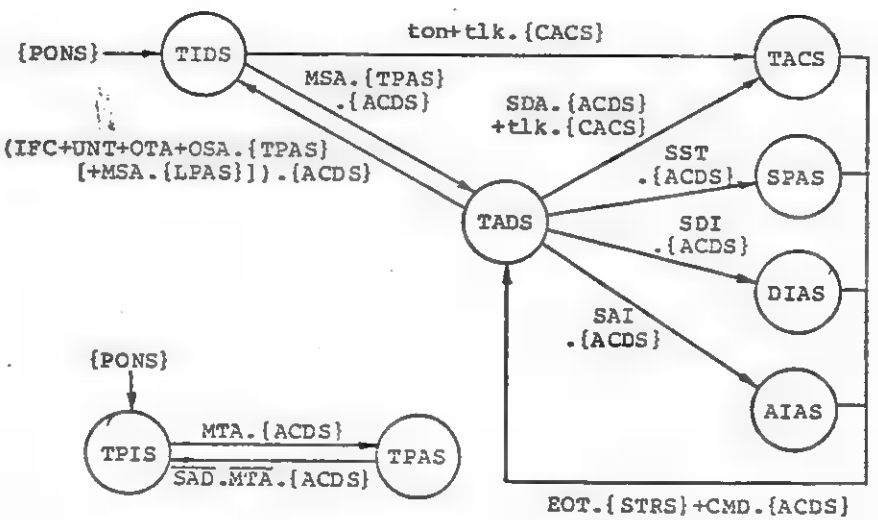


TABLE 2-7: T, TE MNEMONICS

Messages

tlk - local talk
ton - talk only

CMD - command
EOT - end of transmission
IFC - interface clear
MLA - my listen address
MSA - my secondary address
MTA - my talk address
OSA - other secondary address
OTA - other talk address
SAD - secondary address
SAI - send accessory ID
SDA - send data
SDI - send device ID
SST - send status
UNT - untalk

Interface States

AIAS - accessory identify active state (links to R, AH, SH functions)
DIAS - device identify active state (links to R, AH, SH functions)
TACS - talker active state (links to R, AH, SH functions)
TADS - talker addressed state (links to R, AH, C, DD functions)
TIDS - talker idle state
TPAS - talker primary addressed state (links to LE, TE functions)
TPIS - talker primary idle state

ACDS - acceptor data state (from AH function)
CACS - controller active state (from C function)
LPAS - listener primary addressed state (from LE function)
PONS - power on state (from PD function)
STRS - source transfer state (from SH function)
TPAS - talker primary addressed state (from TE function)

to begin sending the device dependent messages. If a CMD which causes the function to become unaddressed is received while in TADS the function returns to TIDS.

TACS (Talker Active State)

In this state the SDA message has been received and the device is enabled to replace this message with a string of data messages (DOE). These transmissions are controlled by the SH (Source Handshake) function. The last message of the string must be the appropriate EOT to indicate to the controller that this

device is finished with its transmission. When the D (Driver) function indicates that it has begun transmitting the EOT the function returns to TADS. If an asynchronous IFC is received the function immediately aborts back to TIDS through TADS.

SPAS (Serial Poll Active State)

In this state the SST message has been received and the device is enabled to replace this message with one or more bytes of device status information followed by an EOT.

DIAS (Device Identify Active State)

In this state the function has received the SDI message and the device is enabled to replace this message with its device ID string followed by an EOT.

AIAS (Accessory Identify Active State)

In AIAS the SAI frame has been received and the device is enabled to replace it with one or more bytes of accessory ID information followed by an EOT.

TPIS (Talker Primary Idle State)

The function is able to respond to its primary talk address in TPIS but must not recognize its secondary address. This is the power on state.

TPAS (Talker Primary Active State)

In TPAS the function has received its primary talk address and is now able to recognize and respond to its secondary address. Receipt of any command other than MSA or MTA (again) will cause return to TPIS.

Remote Messages

This function does not originate any remote messages, but merely enables the device to send device dependent messages under the control of the SH (Source Handshake) interface function to the D (Driver) function for transmission on the loop. If one of TACS, SPAS, DIAS, or AIAS is not active the device may not send device dependent messages.

Additional Requirements and Guidelines

The device may use the END message within the transmitted data string to indicate the end-of-record condition to the listener(s) without terminating the transmission.

In a talk-only device some special protocol must be observed. In a system with a talk-only device and one or more listen-only devices there is no controller. Therefore the talker must not send the EOT message (which would circulate endlessly) but should begin the next message string immediately. Furthermore, after power on, if the talker has not received back its first message frame after a certain amount of time (say, one

second) it should continue to send this message at this slow, device dependent interval until the frame returns completing the loop handshake. Then the talker may commence normal transmission. Without this provision the first frame might "die" at an as yet unpowered listener and the talker would never send the next frame. This permits devices in this type of system to be powered on in any order.

The loop controller may choose to interrupt the talker's data transmission by holding the current DOE message and inserting the NRD (not ready for data) message in its place. The active talker should retransmit this frame and when the controller receives it the previously held data frame is retransmitted. When the talker then receives this data frame it must then send the appropriate EOT. When the device is enabled to resume transmission it will normally continue the data string from the point of interruption. This is only true for data strings; device ID, accessory ID, and status strings must always restart at the beginning when interrupted.

When a DOE returns to the talker the SRQ bit may be set. If the next frame to be sourced is another DOE the talker should see that the SRQ bit in this frame is also set so that the SRQ will reach a controller between the talker and the requesting device on the loop.

To enhance compatibility between the various devices on the loop it is strongly recommended that wherever possible device dependent messages be sent using the ASCII code.

A device's status byte(s) usually consists of a number of bit flags indicating various device dependent conditions. One of the bits is reserved to indicate that this device did request service (1) or that it did not (0). Refer to the SR (Service Request) interface function and the section on remote message coding in this chapter.

It is recommended that the device ID be a string of ASCII characters sent in the following order: two alpha characters representing the company code, a one to five digit model number, a single alpha character model revision, and a carriage return, line feed sequence. Following this the device may send additional frames if desired to indicate such things as options installed in the device or other information to identify or classify the device capabilities.

Certain loop controllers and certain devices have been specially designed to work together such that the loop protocol is entirely transparent to the user. The accessory ID is used as a fast and efficient means for the controller to locate these devices on the loop. The accessory ID consists of a byte(s) assigned to each device with the appropriate accessory capabilities. Refer to the section on remote message coding in this chapter for a complete description of the accessory ID byte(s).

Since a fixed address may make it difficult or impossible to use a device in a particular loop configuration, it is strongly

recommended that a means be provided for the user to change the device's address (and secondary address, if applicable). This could take the form of address switches or the inclusion of one of the automatic address interface functions, for example.

Please note that a device's listen address and talk address may or may not be the same value.

The loop structure and loop handshake are ideal for error checking purposes since the originator of a message can compare the received frame with the original transmitted frame. While not specifically required it is strongly recommended that devices perform this error checking function to enhance loop reliability. After a correct transmission the device sources ETO (end of transmission, OK). If an error is detected, the device would source ETE (end of transmission with error) to signal the controller that a transmission error has occured. These are the two possible EOT messages. If a device does not perform error checking it must only source ETO after the transmission is complete.

If no part of the T or TE function is implemented this capability is designated T0 or TE0 respectively. The designer may choose among several possible subsets in implementing the T or TE function. All subsets (except T0, TE0) must include TIDS and TADS (TE subsets must also include TPIS and TPAS) and one or more of TACS, SPAS, DIAS, and AIAS. These capabilities are designated by the numbers 1, 2, 3, and 4 respectively. If TACS is present the device may operate in the talk-only mode. If this capability is present a switch must be provided to generate the ton local message. If the talk-only mode is not implemented ton must always be sent false. The talk-only capability is designated with the number 5. If the device has the ability to unaddress if its listen address is received the optional term must be implemented in the state diagram. Otherwise this term must be omitted. The presence of this capability is designated by the number 6. Note that this option requires the presence of the L or LE function. To illustrate the use of the capability subset designations, a few examples are in order. A device which can send data and status only would be given T1,2 or TE1,2 designation. The same device with the inclusion of talk-only mode would be listed as T1,2,5 or TE1,2,5. A device that can send only status and its accessory ID and which has unaddress capability would be T2,4,6 or TE2,4,6.

2.9 C (Controller) Interface Function

General Description

The C function provides a device with the ability to send interface messages (including CMD, RDY, and IDY messages) to devices on the loop. It also provides the capability to conduct parallel polls, to detect the SRQ message to determine which devices require service, and to detect transmission errors in device dependent message transmissions. These capabilities exist only while the controller function is in its active state.

FIGURE 2-10: C STATE DIAGRAM

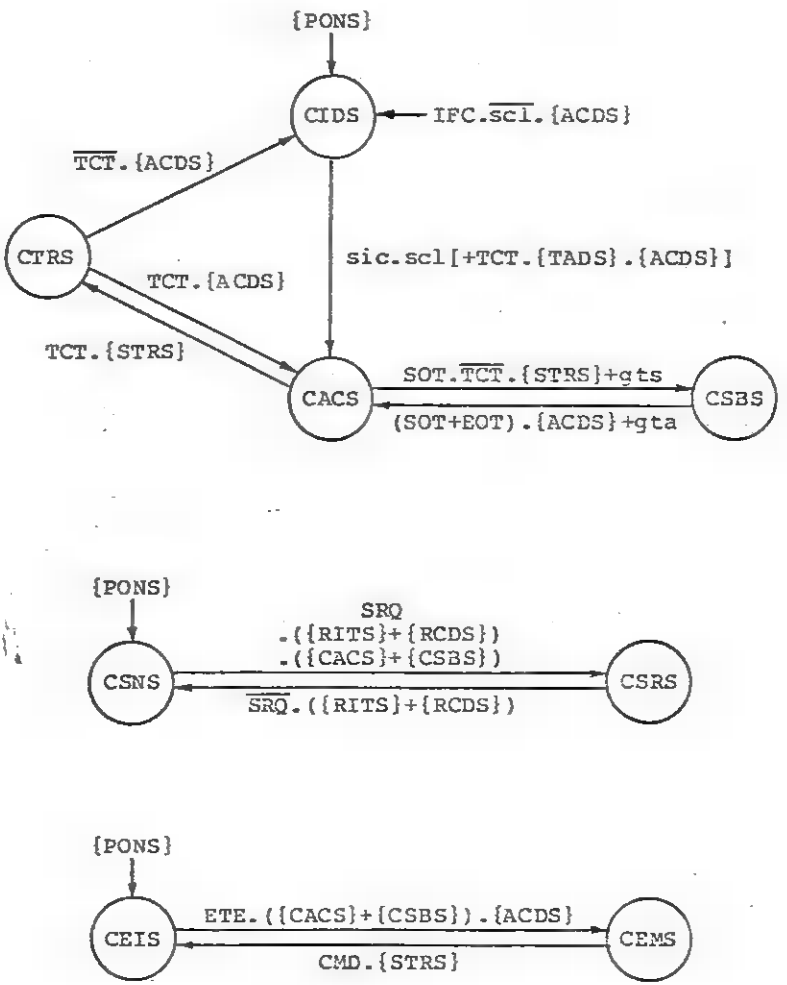


TABLE 2-8: C MNEMONICS

Messages

gta - go to active
gts - go to standby
scl - system control
sic - send interface clear

CMD - command
EOT - end of transmission
ETE - end of transmission with error
IFC - interface clear
SOT - start of transmission
SRQ - service request
TCT - take control

Interface States

CACS - controller active state (links to R, AH, SH, L, LE, T, TE, C, PD functions)
CEIS - controller error idle state
CEMS - controller error mode state
CIDS - controller idle state
CSBS - controller standby state (links to R, AH, C functions)
CSNS - controller service not requested state
CSRS - controller service requested state
CTRS - controller transfer state

ACDS - acceptor data state (from AH function)
CACS - controller active state (from C function)
CSBS - controller standby state (from C function)
PONS - power on state (from PD function)
RCDS - receiver data state (from R function)
RITS - receiver immediate transfer state (from R function)
STRS - source transfer state (from SH function)
TADS - talker addressed state (from T, TE function)

If more than one device which possesses the C function is on the loop, then all but one must be in the idle state at any given time. The device with the active C function is called the controller-in-charge or currently active controller. Protocol is provided to allow controllers to take turns as currently active controller of the loop.

One and only one controller device may send the local message scl true indicating that it is the system controller. This condition should remain true throughout the operation of the interface. The system controller takes charge at power on and is the only device which can source the IFC message at any time regardless whether it is the currently active controller.

CIDS (Controller Idle State)

In CIDS the device relinquishes all loop control capability. The function powers on in this state. When the function receives the TCT message it transitions to CACS, becoming the currently active controller. With the proper local messages, the system controller may take this transition without receiving the TCT. The system controller takes charge of the loop by sending IFC asynchronously.

CACS (Controller Active State)

In this state the device is enabled to send interface messages through the SH (Source Handshake) function to the D (Driver) function for transmission on the loop. If the function enables a device dependent message transfer between a talker and listener(s) on the loop it will transition to CSBS. The controller may also force this transition with the gts local message. If the device is in the process of passing control to another device it will enter CTRS.

CSBS (Controller Standby State)

In CSBS the function has enabled a device dependent message transfer between a talker and listener(s) on the loop by transmitting an SOT message. If the SOT returns or when the function receives the EOT at the end of the device dependent message transfer it resumes active control of the loop. The function may also return active with the gta local message.

CTRS (Controller Transfer State)

In this state the device has sent the TCT message in an attempt to pass control of the loop to another device. If the attempt is unsuccessful the TCT returns and the device must resume control of the loop. If any other frame returns the other device has assumed control and this device should enter CIDS.

CSNS (Controller Service Not Requested State)

In CSNS the function is signaling the device that no device on the loop has requested service. This is the power on state.

CSRS (Controller Service Request State)

In CSRS the function is indicating to the device that at least one device on the loop has requested service. The return to CSNS will only occur when a message is received which could contain an SRQ and the SRQ was false. CMD and RDY messages do not contain SRQ and hence do not cause SRQ to be either true or false.

CEIS (Controller Error Idle State)

In this state the function is indicating to the device that no error has been detected in a device dependent message transfer. If the ETE message is received the function moves to CEMS.

CEMS (Controller Error Mode State)

In CEMS the function is indicating to the device that an error has occured in a device dependent message transfer. When the controller sends its next command the function resets to CEIS.

Remote Messages

This function does not originate any remote messages but enables the device function to originate interface messages under control of the SH (Source Handshake) function for transmission by the D (Driver) function. These are interface control messages as opposed to device dependent messages, and include CMD, RDY, and IDY message frames. The device may only send these messages while CACS is active.

Additional Requirements and Guidelines

The device which sources a device dependent message may not source the next frame until the present frame returns. Likewise, a device which receives a device dependent message may not retransmit it until it is ready to receive the next message. This constitutes the normal loop handshake. Since it is often the case that only one device at a time is to receive these messages anyway, this type of handshake does not represent a speed penalty. In the case of CMD messages which are sourced by the controller device the situation is very different. These messages are quite often intended for all devices on the loop and if each one waited until it was ready for the next message before transmitting the CMD to the next device, loop throughput would suffer greatly. For this reason all devices immediately retransmit CMD messages so they can all interpret the message at almost the same time. When the message returns to the controller, it must source the RFC frame. Each device retransmits the RFC only when it is ready to receive the next CMD. When the RFC returns to the controller, it knows that all devices on the loop are ready to proceed. This CMD-RFC handshake has a significant effect on the loop speed when the controller is sourcing interface messages.

The currently active controller may only source the IDY message asynchronously to the loop handshake. The active controller may send the IDY at any time but will normally use this message after a long delay in loop transmission has ocurred to determine a need for service or to verify loop continuity. The IDY may be sent during a device dependent transmission or after a CMD-RFC sequence the controller has itself sourced. All devices should be implemented so that in either case they will respond properly to the IDY and yet still continue correct response to the previous message. The asynchronous IDY must not disturb normal loop operation. The controller device uses the local messages gta and gts to initiate the asynchronous IDY (as well as the NRD sequence described below).

Devices on the loop indicate a need for service from the controller by setting the SRQ bit in DOE frames. When rapid response is more important the controller may execute a parallel

poll by sending the IDY message. When configured to do so devices respond by modifying one of the assigned eight data bits in the IDY frame. The SRQ bit is also available in the IDY message. Other devices may also have the capability of asynchronously sourcing their own IDY with the SRQ bit set. This mode would be enabled when the controller does not wish to periodically poll the loop for service requests.

For various reasons the controller may choose to interrupt a device dependent message transmission between a talker and listener(s). When the controller does this it holds the next DOE frame which it receives and transmits instead the NRD (not ready for data) frame. When the NRD returns the controller then retransmits the DOE which was held up. The talker will, upon receiving the DOE message, immediately terminate the transmission by sourcing the proper EOT message. When the controller receives the EOT it performs whatever actions are necessary and may or may not choose to continue the device dependent tranmission by again addressing the proper devices and sending the SOT message.

The system controller is the only device on the loop which is permitted to source the IFC message at any time (whether it is the currently active controller or not) and take control of the loop asynchronously. Under this circumstance it is possible to have an extraneous frame on the loop since the source of the frame may be cleared before the frame returns. To prevent this frame from recirculating the system controller, after sourcing the IFC, must destroy any frames which it receives until the IFC returns. Further, no device may source any frame after retransmitting the IFC (and RFC) until specifically commanded to do so by succeeding interface messages. The sic or gta local messages may initiate this sequence depending on the state of the system controller.

When power is first applied the system controller should observe special protocol to insure that all devices on the loop are present and properly connected. Typically the IFC message is sent at regular slow intervals until it returns to the controller indicating loop continuity. The controller may then source the RFC to complete the handshake and commence normal operation.

It is strongly recommended that the controller error check its messages by comparing the received frame with the previously transmitted frame to guarantee proper reception of messages and to enhance reliability.

If a device has no controller capability it is designated C0. Basic controller capability is designated C1 and includes the ability to send interface messages and detect transmission errors by interpreting the ETO or ETE which the talker sources at the end of its transmission. The states CIDS, CACS, CSBS, CEIS, and CEMS are necessary for C1. Several additional capabilities can also be implemented. If the device can be the system controller and send the IFC message at any time the number 2 should be added to the designation. If this capability is not present the sic and scl messages must be always false. It is strongly recommended that system controllers be provided with some manual means of disabling the scl message so that they may

be used in a loop with multiple controllers. The number 3 should be added if the device is capable of responding to the SRQ message. The states CSNS and CSRS are necessary here but must be omitted otherwise. Note that this capability is very rudimentary since the L or LE function is also necessary to allow the device to receive and interpret status bytes. The number 4 represents the ability to pass and receive control of the loop. This requires the state CTRS and the optional term in the CIDS to CACS transition. If this capability is absent CTRS and the optional term must be omitted. The T or TE function must also be present to receive control. If the controller has the ability to configure devices for and execute the parallel poll operation the number 5 shall be added. If the controller can interpret, source, and enable other devices to source asynchronous IDY messages a 6 should be included. Ability to assign automatic addresses is given the number 7. All controllers will, therfore, include C1 with, optionally, some combination of the other capabilities. A simple system controller with only SRQ and auto address capability would be C1,2,3,7. A satellite controller with SRQ and control passing would be C1,3,4.

2.10 DC (Device Clear) Interface Function

General Description

The DC interface function provides the device with the ability to be cleared or initialized. All devices on the loop which implement the capability may be cleared together or only the addressed device(s) may be cleared.

DCIS (Device Clear Idle State)

In this state the DC function is not actively clearing the device function.

DCAS (Device Clear Active State)

In DCAS the the function is signaling the device to clear or initialize its internal function.

Remote Messages

This function does not originate or enable any other function to originate any remote message. It only enables the device to receive and respond to the DCL and SDC messages.

Additional Requirements and Guidelines

Note that this function affects only the device function and has no effect on the other interface functions (which are initialized by the IFC message, if at all). The device may use the DC function for any purpose consistent with its operation. This would typically include placing the device (not the interface) in its power on state, but may be used to place the device function in any predefined state as specified by the designer.

FIGURE 2-11: DC STATE DIAGRAM

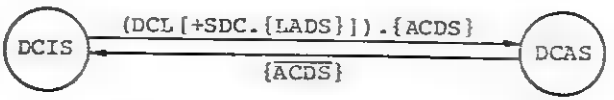


TABLE 2-9: DC MNEMONICS

Messages

DCL - device clear
SDC - selected device clear

Interface States

DCAS - device clear active state
DCIS - device clear idle state

ACDS - acceptor data state (from AH function)
LADS - listener addressed state (from L, LE function)

If the DC function is not implemented the designation is DC0. DC1 designates the ability to respond to the DCL message. All devices with DC1 are cleared together by the DCL message. If the device also includes the ability to respond to the SDC message DC2 is the designation. This message clears only addressed devices. This capability requires the L or LE function and the optional terms in the DC state diagram. If this capability is not implemented the optional terms must be omitted. Repeating, a device which can only respond to DCL is designated DC1 while a device that can respond to both DCL and SDC is designated DC2.

2.11 DT (Device Trigger) Interface Function

General Description

The DT function gives the device the ability to have its basic operation started on command. Since the device must be addressed in order to respond to the trigger command, either a single device or a group of devices may be triggered at the same time. The command does not affect other interface functions.

FIGURE 2-12: DT STATE DIAGRAM

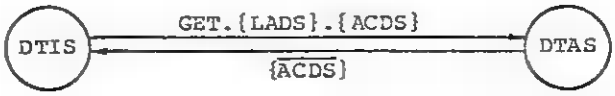


TABLE 2-10: DT MNEMONICS

Messages

GET - group execute trigger

Interface States

DTAS - device trigger active state
DTIS - device trigger idle state

ACDS - acceptor data state (from AH function)
LADS - listener addressed state (from L, LE function)

DTIS (Device Trigger Idle State)

In this state the DT function is not active.

DTAS (Device Trigger Active State)

In DTAS the DT function is signaling the device function to start performing its basic operation.

Remote Messages

This function does not originate or enable any other function to originate any remote message. It only enables the device function to receive and respond to the GET message.

Additional Requirements and Guidelines

It is recommended that the device commence its basic operation as soon as DTAS becomes active. Once the device operation is started the device should not respond to further transitions of the DT interface function until the operation is complete. No capability is designated DT0; complete capability is designated DT1 and requires the L or LE function. No subsets are permitted.

FIGURE 2-13: PP STATE DIAGRAM

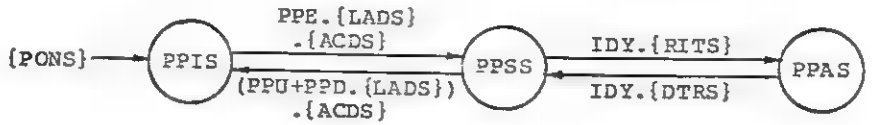


TABLE 2-11: PP MNEMONICS

Messages

IDY - identify
PPD - parallel poll disable
PPE - parallel poll enable
PPU - parallel poll unconfigure

Interface States

PPAS - parallel poll active state
PPIS - parallel poll idle state
PPSS - parallel poll standby state

ACDS - acceptor data state (from AH function)
DTRS - driver transfer state (from D function)
LADS - listener addressed state (from L, LE function)
PONS - power on state (from PD function)
RITS - receiver immediate transfer state (from R function)

2.12 PP (Parallel Poll) Interface Function

General Description

Parallel poll is conducted via the IDY message sourced by the active controller. The PP interface function gives a device the ability to present to the controller one bit of status information without having been addressed to talk. The eight data bits in the IDY message carry this status information from the devices to the controller. This allows up to eight devices to use the parallel poll facility at the same time with a one bit per device assignment. Any number of devices can be accomodated by sharing the available bits. Devices can be configured to present the logical OR of their status or the logical AND of the status on a particular bit. The use of the parallel poll facility requires the commitment of the controller to

periodically conduct the poll by sending the IDY message.

The parallel poll may be used by devices to indicate a need for service. Parallel poll differs from the SRQ message in that all devices use the same bit in SRQ but may use different bits in the IDY. For this reason the serial poll to determine which device requested service may not be necessary with parallel poll. Devices may respond with SRQ anytime a message is present which can accept the SRQ bit, but devices may only respond to parallel poll when solicited by the controller (via the IDY message).

PPIS (Parallel Poll Idle State)

The PP interface function powers on in this state and while in PPIS may not respond to parallel polls conducted by the controller. IDY messages must be retransmitted exactly as received. When the device receives the PPE command it is configured to respond to parallel poll and enters PPSS.

PPSS (Parallel Poll Standby State)

In PPSS the device is configured to respond to parallel polls conducted by the active controller but is not presently doing so because the IDY message is not present. When the IDY is received the function enters PPAS.

PPAS (Parallel Poll Active State)

The function is actively responding to a parallel poll issued by the controller. Depending on the configuration received via the PPE command and the state of the device's individual status bit the received IDY message may be modified before it is retransmitted by the D (Driver) function.

Remote Messages

The PP interface function originates a parallel poll response message while in PPAS only. This is a single bit message which may result in modification of one bit of an IDY frame as it flows through this device. The parallel poll response is determined by configuration information received via the PPE command and by the state of the individual status bit in the device, as well as by the value of the incoming IDY bits.

Additional Requirements and Guidelines

Three bits in the PPE command are reserved to assign the device a bit number on which to respond to parallel polls. If the bits are 000, the device should respond on the least significant bit (D1), and so on in binary sequence 001, 010, 011, etc. to 111, which indicates the most significant bit (D8) as the response bit.

Another bit in the PPE command is reserved to indicate to the device the logical value it must use to show a positive response to parallel poll. If the bit is a 0, the device's positive response is to not modify the incoming bit; the

negative response would be to OR in a 1 in the assigned bit. If the PPE bit is a 1, the positive response is to OR in a 1, the negative response is no modification. This permits the controller to configure devices on a single bit such that the result is the logical OR or the logical AND of the various devices' responses.

If this function is not implemented the designation is PP0; if it is implemented the designation is PP1, which requires the L or LE function. No subsets are permitted.

2.13 SR (Service Request) Interface Function

General Description

The SR function provides the device the ability to request service from the controller. Devices request service by setting the SRQ (C1) bit in DOE and IDY messages. The set SRQ bit notifies the controller that one or more devices on the loop has requested service. CMD and RDY frames do not have an SRQ bit and may not transmit the service request.

SRIS (Service Request Idle State)

In SRIS the SR function is not requesting service. The function powers on in this state. When the rsv local message goes true the function enters SRSS provided the controller is not currently requesting the device's status byte(s) via a serial poll.

SRSS (Service Request Standby State)

In this state the device is ready to request service but is not actively engaged in doing so. When a DOE or IDY frame arrives the function transitions to SRAS. If the device has been enabled to respond asynchronously and the arq local message is true the transition is to ARSS. If the device no longer requires service (even though the controller has not responded) the function returns to SRIS. If the controller conducts a serial poll to retrieve status or determine which device requested service the function will enter SRHS.

SRAS (Service Request Active State)

In SRAS a DOE or IDY message has been received and the device is setting the SRQ bit in this message prior to its transmission by the D (Driver) interface function. When the transmission begins, the function returns to SRSS.

ARSS (Asynchronous Request Source State)

In this state the device sources its own IDY message with the SRQ bit set. When the D function indicates that transmission has begun the function returns to SRSS.

FIGURE 2-14: SR STATE DIAGRAM

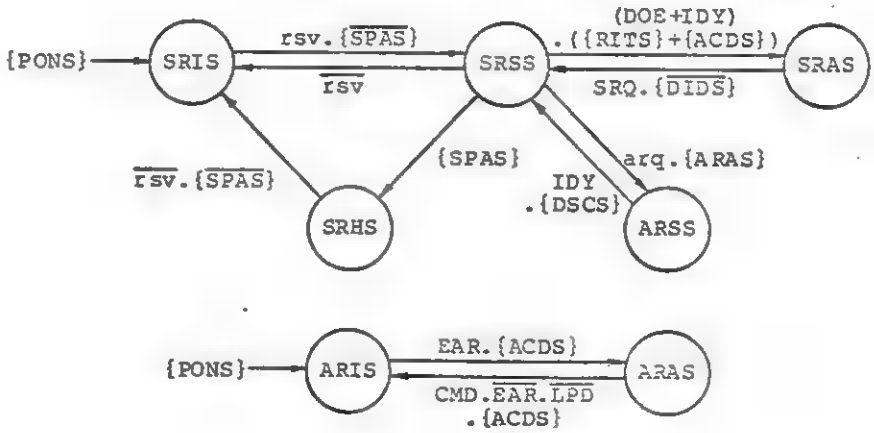


TABLE 2-12: SR MNEMONICS

Messages

rsv - request service
arq - asynchronous request

DOE - data or end
CMD - command
EAR - enable asynchronous request
IDY - identify
LPD - loop power down
SRQ - service request

Interface States

ARAS - asynchronous request active state (links to SR function)
ARIS - asynchronous request idle state
ARSS - asynchronous request source state (links to SH function)
SRIS - service request idle state
SRSS - service request standby state
SRAS - service request active state
SRHS - service request hold state

ACDS - acceptor data state (from AH function)
ARAS - asynchronous request active state (from SR function)
DIDS - driver idle state (from D function)
PONS - power on state (from PD function)
RITS - receiver immediate transfer state (from R function)
SPAS - serial poll active state (from T, TE function)

SRHS (Service Request Hold State)

In this state the device requires service but may not request it over the loop. A serial poll has been conducted so that the controller is aware of the device's need for service. When the controller has serviced the device or the need for service no longer exists for some other reason, the device sends the local message rsv false and the function returns to SRIS.

ARIS (Asynchronous Request Idle State)

In ARIS the device may not generate asynchronous service requests. This is the power on state. When the EAR message is received the function moves to ARAS.

ARAS (Asynchronous Request Active State)

In this state the device is enabled to generate asynchronous service requests. The transition from SRSS to ARSS is enabled by this state. If any command other than EAR or LPD is received the function returns to ARIS.

Remote Messages

The SR function originates the SRQ message while in SRAS. This will result in the setting of the SRQ bit in DOE and IDY frames received over the loop and retransmitted by the D (Driver) interface function. The ARSS state enables the SH (Source Handshake) function which in turn enables the device to generate its own IDY with the SRQ bit set.

Additional Requirements and Guidelines

The SR function is intended to be used for a single reason for requesting service within the device. If more than one reason for requesting service exists within a device, a separate SR function and corresponding rsv message should be implemented for each of the reasons.

The state of the rsv message is directly reflected by the bit in the device's status byte which is reserved to indicate to the controller during serial poll that this device did request service. After the serial poll the device may no longer actively request service by setting the SRQ bit but the bit in the status byte will remain set (as will rsv) until the reason for requesting service is no longer present. If more than one reason for service exists, this status bit should be the logical OR of the service requests within that device.

There are three ways in which devices may request service from the active controller. The use of the parallel poll bits and the use of the SRQ bit in DOE and IDY frames requires a commitment on the part of the active controller to periodically poll the loop for service by sending IDY messages. If this is not desireable for one reason or another, the third method of requesting service, asynchronous requests, may be used. When the controller finishes its activities, it sends the EAR command (and RFC, of course) and then allows the loop to go idle. When the

controller receives an asynchronous IDY or decides on its own that it needs to perform further operations, it will source its next command which will disable further asynchronous requests and cause the loop to operate in normal synchronous manner once again. While ARAS is active, the controller may only source IDY and CMD-RFC messages. Since the asynchronous IDY may not have travelled all the way around the loop in this mode, the parallel poll bits will not necessarily be correct.

SR0 indicates no service request capability. Basic service request capability is indicated by SR1. The states ARSS, ARIS, and ARAS are omitted. SR1 also requires T2 or TE2 in order to send the status byte(s). SR2 indicates full SR capability; no states are omitted; T2 or TE2 is required. SR2 provides both basic and asynchronous service requests. Note that the arq message may be sent true only in SRSS; It must be sent false immmediately on entry to ARSS.

2.14 AA (Automatic Address) Interface Function

General Description

The AA function gives the device the ability to have its address assigned by a command on the loop. If all devices on the loop have this capability, the user is relieved of the necessity of manual address configuration. The AA function permits a device to accept a one frame address assignment. This permits a maximum of 31 devices on the loop. There are two other variations of automatic addressing described in this document. Only one of the three need be implemented in any given device.

AAUS (Auto Address Unconfigured State)

In this state the function has not yet been configured and may only respond to a preset address or to address switches if the designer has provided one of these other methods of addressing. The AA function powers on in AAUS.

AAIS (Auto Address Increment State)

In AAIS the AAD command containing this device's assigned address has been received and the device is in the process of incrementing this address by one for the next device. When transmission of NAA commences the function enters AACS.

AACS (Auto Address Configured State)

The device is now configured and must only respond to the address assigned in the AAD message. If the AAU command is received the function returns to AAUS.

Remote Messages

This function originates the NAA message by incrementing the AAD message as it passes from the R (Receiver) function to the D (Driver) function. This takes place only in AAIS.

FIGURE 2-15: AA STATE DIAGRAM

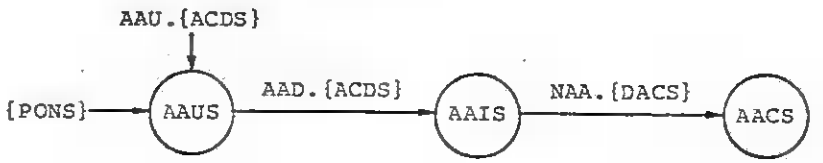


TABLE 2-13: AA MNEMONICS

Messages

AAU - auto address unconfigure
AAD - auto address
NAA - next auto address

Interface States

AACS - auto address configured state
AAIS - auto address increment state
AAUS - auto address unconfigured state

ACDS - acceptor data state (from AH function)
PONS - power on state (from PD function)
STRS - source transfer state (from SH function)

Additional Requirements and Guidelines

The least significant five bits of the AAD message contain the binary address assignment, addresses 0 through 30. Address 31 is not a legal address (11111). This message is called IAA (illegal auto address) and the AA function does not respond in any way to this message though it may generate this message if the incoming AAD contained address 30.

Typically the controller will configure the loop by sending AAD with address 0 or 1. Devices will each in turn accept the address and increment it for the next device. The value which returns to the controller indicates the number of devices on the loop. The controller may choose to accept the address which returns as its own address also. It is strongly recommended that the controller have its own address and this is necessary if multiple controllers are used on the loop.

If the message that returns to the controller is IAA (illegal auto address) the loop may contain exactly the maximum number of devices or it may contain too many devices. At this

point the controller could send AAD with address 30 and if it returned unmodified the controller would proceed knowing the loop contained exactly the maximum numoer of devices. If IAA returned again the controller would know there were too many devices and should signal an error to the user. The loop will not function properly with more than one device responding to the same address.

No AA function capability is designated AA0; full capability is AA1. No subsets are permitted.

2.15 AE (Auto Extended Address) Interface Function

General Description

The AE function provides capability similar in nature to the AA (Automatic Address) function but it permits a device to receive a two frame address assignment. This permits a maximum of 961 devices on the loop.

AEUS (Auto Extended Unconfigured State)

In this state the function has not been configured and may only respond to a preset address or to address switches if the designer has provided one of these other methods of addressing. The function powers on in AEUS.

ASIS (Auto Secondary Increment State)

In this state the device has received its assigned secondary address (AES) and is in the process of incrementing this address by one for the next device. When transmission of NES begins the function enters AWPS.

AWPS (Auto Wait for Primary State)

In AWPS the device is waiting to receive its primary address assignment via the AEP command. When AEP is received the function enters AECS.

AECS (Auto Extended Configured State)

In this state the device has received both its assigned secondary and primary addresses and must only respond to these addresses. If the AAU message is received the function returns to AEUS.

Remote Messages

This function originates the NES message by incrementing the AES message as it passes from the R (Receiver) function to the D (Driver) function. This only occurs in ASIS.

Additional Requirements and Guidelines

As with one frame auto addressing, the least significant

five bits of the AES and AEP contain the address assignment for the secondary and primary extended address. Address 31 is illegal for both the secondary and primary address and is represented by the IES (illegal extended secondary) and IEP (illegal extended primary) messages, respectively.

FIGURE 2-16: AE STATE DIAGRAM

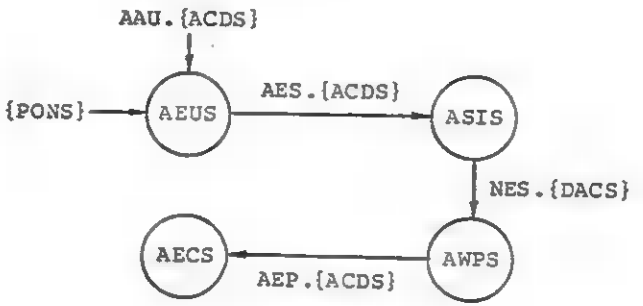


TABLE 2-14: AE MNEMONICS

Messages

AAU - auto address unconfigure
AEP - auto extended primary
AES - auto extended secondary
NES - next extended secondary

Interface States

AECS - auto extended configured state
AEUS - auto extended unconfigured state
APAS - auto primary addressed state
ASIS - auto secondary increment state
AWPS - auto wait for primary state

ACDS - acceptor data state (from AH function)
PONS - power on state (from PD function)
STRS - source transfer state (from SH function)

In a typical sequence the controller would first send the AES message. The first group of devices would receive their secondary addresses and increment the AES until IES is generated. No other devices will respond to IES and it will

return to the controller. The controller will then send the first extended primary address. Only those devices which received legal secondary addresses will respond to and accept this primary address. Note that the primary address is not incremented. This group of devices is now configured and will no longer respond to AES or AEP commands (AAU would unconfigure all devices). The controller now sends out another AES command to configure the second group of devices in the same manner. This should continue until the AES returns to the controller with a legal address. The controller then knows how many devices are on the loop.

No AE capability is designated AE0; full capability is AE1. No subsets are permitted.

2.16 AM (Auto Multiple Address) Interface Function

General Description

The AM function provides much the same capability as the AE (Auto Extended Address) function in that a two frame address is assigned by commands, however the AM function is intended for use in devices which have multiple functions such that the device is assigned the primary address and each function within that device is assigned a different secondary address. This allows for a maximum of 31 devices each containing a maximum of 31 functions.

AMUS (Auto Multiple Unconfigured State)

In this state the function has not yet been configured and may only respond to preset addresses or to address switches if the designer has provided one of these other means of addressing. The AM function powers on in AMUS.

APIS (Auto Primary Increment State)

In this state the function has been assigned its primary address via the AMP command and is in the process of incrementing this address by one for the next multiple function device. All functions within this device respond to this primary address. When transmission of NMP begins the function enters AWSS.

AWSS (Auto Wait for Secondary State)

The function has transmitted the NMP message and is waiting to receive the ZES message.

AMIS (Auto Multiple Increment State)

In this state the device has received the ZES message (auto extended secondary address 0) and is assigning each function of the device a secondary address beginning with zero. The ZES message is incremented by the number of functions in the device and when transmission of this NES message begins the AM function enters AMCS.

AMCS (Auto Multiple Configured State)

In AMCS each function of the device has been assigned the device's primary address and an individual secondary address and must only respond to these addresses. If the AAU message is received the AM function returns to AMUS.

FIGURE 2-17: AM STATE DIAGRAM

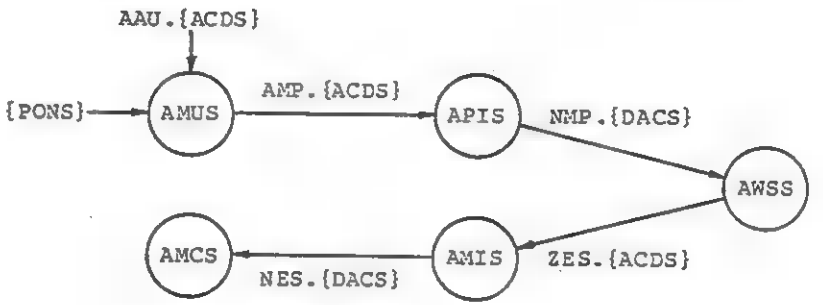


TABLE 2-15: AM MNEMONICS

Messages

AAU - auto address unconfigure
AMP - auto multiple primary
NES - next extended secondary
NMP - next multiple primary
ZES - zero extended secondary

Interface States

AMCS - auto multiple configured state
AMIS - auto multiple increment state
AMUS - auto multiple unconfigured state
APIS - auto primary increment state
AWSS - auto wait for secondary state

ACDS - acceptor data state (from AH function)
PONS - power on state (from PD function)
STRS - source transfer state (from SH function)

Remote Messages

The AM function originates the NMP message by incrementing the AMP message by one as it passes from the R (Receiver) function to the D (Driver) function. This only occurs in APIS. This function also originates the NES message by incrementing the ZES message as it passes from the R (Receiver) function to the D (Driver) function. This occurs only in AMIS and the increment value depends on the number of functions in the device.

Additional Requirements and Guidelines

In the typical sequence here, the controller issues first the AMP message. The return address in the message indicates the number of devices on the loop (IMP, illegal multiple primary, would indicate either the maximum number or too many devices on the loop, as in simple auto addressing). Then the controller would send ZES once for each device. The return values indicate the number of separate functions in each device.

AM0 indicates no AM capability; AM1 indicates full capability. No subsets are permitted.

2.17 RL (Remote Local) Interface Function

General Description

The RL interface function provides a device with the capability to select between two sources of input information. The function indicates to the device that either input from the front panel controls (local) or corresponding input from the interface (remote) is to be used.

RIDS (Remote Idle State)

In RIDS the function is indicating to the device that the active controller has not placed the interface in remote mode. The function powers on in this state. If the REN message is received the function enters RACS.

RACS (Remote Active State)

In this state the function is indicating that the active controller has placed the interface in remote mode, however, device functions remain under local control. If the NRE message is received the function returns to RIDS.

LOCS (Local State)

In LOCS all local controls of the associated device functions (both front and rear panel) are operative. The device may store but not respond to corresponding device dependent messages from the interface. The RL function powers on in this state. If MLA is received the function will enter REMS, provided the rtl local message is not true. If LLO is received the function enters LWLS.

FIGURE 2-18: RL STATE DIAGRAM

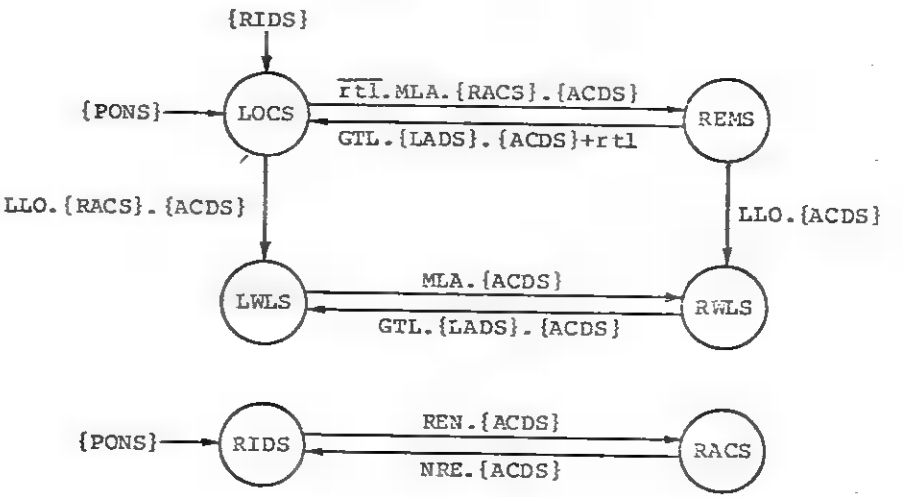


TABLE 2-16: RL MNEMONICS

Messages

rtl - return to local

GTL - go to local
LLO - local lockout
MLA - my listen address
NRE - not remote enable
REN - remote enable

Interface States

LOCS - local state
LWLS - local with lockout state
RACS - remote active state (links to RL function)
REMS - remote state
RIDS - remote idle state (links to RL function)
RWLS - remote with lockout state

ACDS - acceptor data state (from AH function)
LADS - listener addressed state (from L, LE function)
PONS - power on state (from PD function)
RACS - remote active state (from RL function)
RIDS - remote idle state (from RL function)

LWLS (Local With Lockout State)

In this state local controls are operative. The device may store but not respond to corresponding device dependent messages from the loop. If MLA is received the function enters RWLS.

REMS (Remote State)

In REMS the local controls which have corresponding remote controls are inoperative and those device functions are under control of messages received over the interface. This does not include those controls which generate local messages for the interface functions of this device. If the GTL message is received or if the local message rtl goes true the function returns to LOCS; if the LLO message is received the function enters RWLS.

RWLS (Remote With Lockout State)

In this state, as in REMS, the local controls are inoperative, however the function will not now respond to the rtl local message. If the GTL message is received the function goes to LWLS.

Remote Messages

This interface function does not originate or enable any other function to originate remote messages.

Additional Requirements and Guidelines

Note that this function does not affect a device's ability to send or receive messages on the loop, though in LOCS and LWLS the device functions may not respond. Selected devices may be placed in remote control while others are left under local control by sending the device's listen address.

When the function transitions from one of the local states to one of the remote states the device functions may retain their local settings until receiving specific remote settings, or they may immmediately revert to the remote settings previously received. Conversely, when the function makes the transition from remote to local, the device functions may retain their remote settings until local controls override them, or they may immediately revert to the present settings of the local controls.

A means of controlling the rtl message must be provided, for example, a pushbutton. The rtl message must not always be true.

If a device has no RL capability it is designated RL0. Basic RL capability is designated RL1 and requires L or LE capability. LWLS and RWLS are omitted in RL1 since it does not include local lockout capability. RL2 indicates full RL capability including local lockout. All states must be implemented; the L or LE function is required.

FIGURE 2-19: PD STATE DIAGRAM

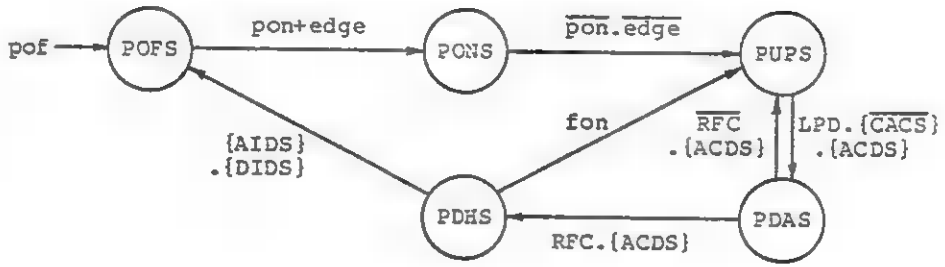


TABLE 2-17: PD MNEMONICS

messages

fon - force on
pof - power off
pon - power on

edge - wake-up pulse on loop

LPD - loop power down
RFC - ready for command

Interface States

PDAS - power down active state
PDHS - power down hold state
PONS - power on state (links to most interface functions)
POFS - power off state (implicitly links to most interface functions)
PUPS - power up state

ACDS - acceptor data state (from AH function)
AIDS - acceptor idle state (from AH function)
CACS - controller active state (from C function)
DIDS - driver idle state (from D function)

2.18 PD (Power Down) Interface Function

General Description

This function gives a device the capability to be placed in a power down or low power mode by command received over the loop and likewise to be reactivated for normal operation. When in the power down mode the device only needs to monitor the loop for any pulse. When any frame arrives the pseudomessage edge is

generated which reactivates the device and places the interface functions in the power on state.

POFS (Power Off State)

This state is entered anytime the local message pof is true. This message is normally generated from the off position of the device power switch. Most other interface functions also have the equivalent of this state though it is not shown for clarity. When POFS becomes active in the PD function, it causes an immediate transition to this equivalent state in all other interface functions. When the device power switch generates the pon local message or a frame arrives generating the edge message, the function enters PONS.

PONS (Power On State)

PONS causes the entry to the first (power on) state in most interface functions. The pon or edge message must remain true for sufficient time to allow interface and device functions to properly respond. When these messages both are false again, the function enters PUPS, the normal operating state.

PUPS (Power Up State)

PUPS is the normal operating state for the device. If the LPD command is received and the device is not the active controller, the function enters PDAS. The active controller must remain on in order to receive asynchronous requests from devices which may wake up the loop, or to be able to wake up the loop itself.

PDAS (Power Down Active State)

In PDAS the device is waiting for the RFC following the LPD command. When the RFC is received the function enters PDHS. If any other remote message is received the function aborts and returns to PUPS.

PDHS (Power Down Hold State)

In this state the function is waiting for the completion of the retransmission of the RFC message before actually going to the power down mode, POFS. If the fon local message is true, the function will return to PUPS.

Remote Messages

This function does not originate or enable any other function to originate any remote message.

Additional Requirements and Guidelines

The active controller should wake up a powered down loop by repeatedly sending a command (NUL is recommended) until it returns and then sending the RFC to complete the handshake. This will permit devices whatever time is necessary to complete their individual power on sequences and begin to recognize and

retransmit frames. The controller would do this either on its own initiative or in response to an asynchronous IDY if this mode was enabled prior to power down.

Devices may use a three position power switch to enable or disable the power down function. The three positions would be: "off", "standby" (fon false), and "on" (fon true). The fon message should also be used by devices needing to remain on to wake up the loop later.

All devices must implement at least POFS, PONS, and PUPS. This capability is designated PD0. The edge message must always be sent false. PD0 devices simply ignore the LPD command. PD1 indicates full capability. All states must be implemented as shown. No other subsets are permitted.

2.19 DD (Device Dependent Command) Interface Function

General Description

A number of command codes are specifically reserved for the device designer to use as he sees fit. There are 32 DDL commands; the device must be listener addressed in order to respond. There are also 32 DDT commands; the device must be talker addressed to respond.

DDIS (Device Dependent Idle State)

In this state the device is not performing the operation the designer has specified for the command represented by this interface function. When the command is received and the device is appropriately addressed, the function transitions to DDAS.

DDAS (Device Dependent Active State)

In DDAS the function is indicating to the device that it should begin performing the operation specified for the command represented by this interface function.

Remote Messages

This function does not originate or enable any other function to originate any remote message. It only enables the device function to respond to the particular device dependent command represented by this interface function.

Additional Requirements and Guidelines

It is recommended that the device commence its response as soon as DDAS becomes active. Once this response is begun the device should not respond to further transitions of the DD interface function until the operation is complete.

A separate DD interface function should be implemented for each device dependent command to which the device responds. Only the appropriate one of the two optional terms in the transition

will be implemented in any particular DD function.

No capability is designated DD0. If the device responds to one or more DDL or DDT commands, it should be designated DD1. DD1 devices must include appropriate documentation on each device dependent command response.

FIGURE 2-20: DD STATE DIAGRAM

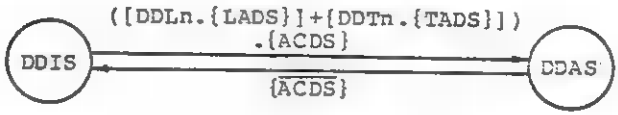


TABLE 2-18: DD MNEMONICS

Messages

DDLn - device dependent listener command n
DDTn - device dependent talker command n

Interface States

DDAS - device dependent active state
DDIS - device dependent idle state

ACDS - acceptor data state (from AH function)
LADS - listener addressed state (from L, LE function)
TADS - talker addressed state (from T, TE function)

2.20 Remote Message Coding

This section gives definitions and codes for all remote messages. Remote messages on the signal lines are sensed by the decoder, are decoded, and are passed on to the interface functions for appropriate action. Interface functions originate and control the transmission of remote messages to the encoder, which encodes and transmits them over the signal lines.

Throughout this section, 0 represents a logical zero signal, 1 represents a logical one signal, and X represents a receiver "don't care" or a transmitter "no change" for retransmission.

The first three bits of each frame (C3, C2, C1 in that order) are called control bits and serve to classify frames into major categories for retransmission and response. The following table gives the codes for these bits of the frame and their meanings. Please note that the interface system responds in precisely the same way to both data and end messages. The end bit is provided solely as a mechanism for devices to use to indicate an end-of-record condition within device dependent message transmissions without necessarily terminating the transmission. The end bit is C2 and the service request bit is C1 in those frames which have provision for these bits.

TABLE 2-19: CONTROL BIT CODES

C3 C2 C1	Classification	Mnemonic
0 0 0	data byte	DAB
0 0 1	data byte with service request	DAB (SRQ)
0 1 0	end byte	END
0 1 1	end byte with service request	END (SRQ)
1 0 0	command	CMD
1 0 1	ready	RDY
1 1 0	identify	IDY
1 1 1	identify with service request	IDY (SRQ)

The remaining eight bits (D8 through D1 in order) are called data bits (not to be confused with a data message) and indicate the specific frame within the general classifications described above. The specific codes for command, ready, and identify messages are given within this section. The codes for data or end messages may be anything which the involved devices are all able to interpret. It is recommended for maximum compatibility, however, that the ASCII code be used whenever possible. Bit 1 of the ASCII code representation corresponds to PIL bit D1 and bit 8 corresponds to D8. If the ASCII code is not used it is recommended that the most significant bit of the data correspond to D8 and the least significant bit correspond to D1.

Messages may be defined as the logical combination (AND, OR, or NOT) of other messages. Messages are therefore divided into several classes and sub-classes. To facilitate the user's understanding of these message relationships a frame hierarchy table is included. This table graphically shows which messages are included within other messages. An alphabetic listing of all messages, their mnemonics, and their coding follows the frame hierarchy table. For convenience, a table of messages by coding is also included.

TABLE 2-20: FRAME HIERARCHY TABLE

```
DOE .... DAB
       . DAB(SRQ)
       . END
       . END(SRQ)

CMD .... ACG .... NUL
       .        . GTL
       .        . SDC
       .        . PPD
       .        . GET
       .        . PPE(0-15)
       .        . DDL(0-31)
       .        . DDT(0-31)
       .
       . UCG .... LLO
       .        . DCL
       .        . PPU
       .        . EAR
       .        . IFC
       .        . REN
       .        . NRE
       .        . AAU
       .        . LPD
       .
       . LAG .... LAD(0-30)
       .        . MLA(0-30)
       .        . UNL(31)
       .
       . TAG .... TAD(0-30)
       .        . MTA(0-30)
       .        . OTA(0-30)
       .        . UNT(31)
       .
       . SAG .... SAD(0-30)
                . MSA(0-30)
                . OSA(0-30)

IDY .... IDY
       . IDY(SRQ)

RDY .... RFC
       .
       . ARG .... EOT .... ETO
       .        .        . ETE
       .        .
       .        . NRD
       .        .
       .        . SOT .... SDA
       .                 . SST
       .                 . SDI
       .                 . SAI
       .                 . TCT
       .
       . AAG .... AAD(0-30)
                . NAA(1-31)
                . IAA(31)
                .
                . AEP(0-30)
                . IEP(31)
                .
                . ZES(0)
                . AES(0-30)
                . NES(1-31)
                . IES(31)
                .
                . AMP(0-30)
                . NMP(1-31)
                . IMP(31)
```

TABLE 2-21: MESSAGE TABLE

MNEMONIC	MESSAGE	CLASS	GROUP	CCC DDDDDDDD 321 87654321	NOTES
AAD	auto address 0-30	RDY	AAG	101 100AAAAA	1
AAG	auto address group	RDY		101 100XXXXX	
AAU	auto address unconfigure	CMD	UCG	100 10011010	
ACG	addressed command group	CMD		100 X000XXXX or 100 101XXXXX or 100 110XXXXX	
AEP	auto extended primary 0-30	RDY	AAG	101 101AAAAA	1
AES	auto extended secondary 0-30	RDY	AAG	101 110AAAAA	1
AMP	auto multiple primary 0-30	RDY	AAG	101 111AAAAA	1
ARG	addressed ready group	RDY		101 01XXXXXX	
CMD	command			100 XXXXXXXX	
DAB	data byte	DOE		00X XXXXXXXX	
DCL	device clear	CMD	UCG	100 00010100	
DDL	device dependent listener command 0-31	CMD	ACG	100 101XXXXX	
DDT	device dependent talker command 0-31	CMD	ACG	100 110XXXXX	
DOE	data or end			0XX XXXXXXXX	
EAR	enable asynchronous requests	CMD	UCG	100 00011000	
END	end	DOE		01X XXXXXXXX	
EOT	end of transmission	RDY	ARG	101 0100000X	
ETE	end of transmission, error	RDY	ARG	101 01000001	
ETO	end of transmission, OK	RDY	ARG	101 01000000	
GET	group execute trigger	CMD	ACG	100 00001000	
GTL	go to local	CMD	ACG	100 00000001	
IAA	illegal auto address	RDY	AAG	101 10011111	

MNEMONIC	MESSAGE	CLASS	GROUP	CCC 321	DDDDDDDD 87654321	NOTES
IDY	identify			11X	XXXXXXXX	
IEP	illegal extended primary	RDY	AAG	101	10111111	
IES	illegal extended secondary	RDY	AAG	101	11011111	
IFC	interface clear	CMD	UCG	100	10010000	
IMP	illegal multiple primary	RDY	AAG	101	11111111	
LAD	listen address 0-30	CMD	LAG	100	001AAAAA	1
LAG	listen address group	CMD		100	001XXXXX	
LLO	local lockout	CMD	UCG	100	00010001	
LPD	loop power down	CMD	UCG	100	10011011	
MLA	my listen address 0-30	CMD	LAG	100	001MMMMM	2
MSA	my secondary address 0-30	CMD	SAG	100	011MMMMM	2
MTA	my talk address 0-30	CMD	TAG	100	010MMMMM	2
NAA	next auto address 1-31	RDY	AAG	101	100NNNNN	3
NES	next extended secondary 1-31	RDY	AAG	101	110NNNNN	3
NMP	next multiple primary 1-31	RDY	AAG	101	111NNNNN	3
NRD	not ready for data	RDY	ARG	101	01000010	
NRE	not remote enable	CMD	UCG	100	10010011	
NUL	null command	CMD	ACG	100	00000000	
OSA	other secondary address	CMD	SAG	100	011TTTTT	4
OTA	other talk address	CMD	TAG	100	010TTTTT	4
PPD	parallel poll disable	CMD	ACG	100	00000101	
PPE	parallel poll enable 0-15	CMD	ACG	100	1000SBBB	5
PPU	parallel poll unconfigure	CMD	UCG	100	00010101	
RDY	ready			101	XXXXXXXX	
REN	remote enable	CMD	UCG	100	10010010	

MNEMONIC	MESSAGE	CLASS	GROUP	CCC DDDDDDDD 321 87654321	NOTES
RFC	ready for command	RDY		101 00000000	
SAD	secondary address 0-30	CMD	SAG	100 011AAAAA	1
SAG	secondary address group	CMD		100 011XXXXX	
SAI	send accessory ID	RDY	ARG	101 01100011	
SDA	send data	RDY	ARG	101 01100000	
SDC	selected device clear	CMD	ACG	100 00000100	
SDI	send device ID	RDY	ARG	101 01100010	
SOT	start of transmission	RDY	ARG	101 01100XXX	
SRQ	service request	DOE or IDY	or	0X1 XXXXXXXX or 111 XXXXXXXX	
SST	send status	RDY	ARG	101 01100001	
TAD	talk address 0-30	CMD	TAG	100 010AAAAA	1
TAG	talk address group	CMD		100 010XXXXX	
TCT	take control	RDY	ARG	101 01100100	
UCG	universal command group	CMD		100 X001XXXX	
UNL	unlisten	CMD	LAG	100 00111111	
UNT	untalk	CMD	TAG	100 01011111	
ZES	zero extended secondary	RDY	AAG	101 11000000	

NOTES

1. AAAAA represents a five bit binary device address in the range 0 to 30. The address may be either a primary listen or talk address or a secondary address, depending on the particular message. The least significant bit corresponds to bit D1.

2. MMMMM represents the particular device address which matches the address assigned to this device.

3. NNNNN represents the incremented device address which is retransmitted by a device during auto address configuration. Normally the increment is one, except in the case of NES for auto

multiple addressing which is one more than the number of secondary addresses reserved by the device. The range of this incremented value is 1 to 31. Note that 31 is not a valid address.

4. TTTTT represents a device address which does not match the address assigned to this device. The address may be either a primary talk address or a secondary address, depending on the particular message.

5. S represents the sense of a positive response to a parallel poll (IDY). If S is 0 the device should do nothing if it requires service and should change its assigned bit to a 1 if it does not require service. If S is a 1 the device should change its assigned bit to a 1 if it requires service and should do nothing if it does not. "Do nothing" means retransmit the frame as received. BBB indicates which bit the device should respond on. 000 means bit D1, 001 means bit D2, and so on to 111, which means respond on bit D8.

6. Coding of the data bits in device dependent messages (DOE) is discussed in the text with two exceptions, status bytes and accessory ID bytes.

A device's status usually consists of a single byte but additional bytes may be sent as necessary. Bit D7 of the first byte is always reserved to indicate that this device is requesting service (1) or is not (0). Bits D8 and D6 to D1 of the first byte and all bits of any additional bytes may be used as desired by the device designer. Devices which have an accessory ID are required to adhere to the following convention and it is recommended that other devices do the same though they need not if circumstances dictate otherwise. If bit D8 is 0, the status byte(s) are as described above. If bit D8 is a 1, bits D6 to D1 represent a binary coded status message number in the first byte. Other bytes are designer specified. The status message in the low order six bits is intended to be a system defined condition of general significance. A table of presently defined system status messages is found in one of the appendices.

Accessory ID is also generally only one byte. The high order four bits (D8 to D5) contain a binary coded device class number indicating the general category of the device, for example, printer, mass storage, etc. The low order four bits (D4 to D1) contain a binary coded number representing the specific device within the general category. Hence, each device which responds with an accessory ID is assigned a unique byte. Accessory ID gives the controller the ability to locate a capability on the loop without caring about a particular device model number. It also provides very simple controllers with limited memory a rapid, efficient way of identifying a particular device when necessary. A table of presently defined accessory ID class numbers and specific devices is found in one of the appendices.

TABLE 2-22: CODING CHART

DDDDD 87654 \ DDD 321	000	001	010	011	100	101	110	111	GROUP
00000	NUL	GTL	.	.	SDC	PPD	.	.	ACG
00001	GET	.	.	.	.	.	.	.	ACG
00010	.	LLO	.	.	DCL	PPU	.	.	UCG
00011	EAR	.	.	.	.	.	.	.	UCG
00100	LAD0	LAD1	LAD2	LAD3	LAD4	LAD5	LAD6	LAD7	LAG
00101	LAD8	LAD9	LAD10	LAD11	LAD12	LAD13	LAD14	LAD15	LAG
00110	LAD16	LAD17	LAD18	LAD19	LAD20	LAD21	LAD22	LAD23	LAG
00111	LAD24	LAD25	LAD26	LAD27	LAD28	LAD29	LAD30	UNL	LAG
01000	TAD0	TAD1	TAD2	TAD3	TAD4	TAD5	TAD6	TAD7	TAG
01001	TAD8	TAD9	TAD10	TAD11	TAD12	TAD13	TAD14	TAD15	TAG
01010	TAD16	TAD17	TAD18	TAD19	TAD20	TAD21	TAD22	TAD23	TAG
01011	TAD24	TAD25	TAD26	TAD27	TAD28	TAD29	TAD30	UNT	TAG
01100	SAD0	SAD1	SAD2	SAD3	SAD4	SAD5	SAD6	SAD7	SAG
01101	SAD8	SAD9	SAD10	SAD11	SAD12	SAD13	SAD14	SAD15	SAG
01110	SAD16	SAD17	SAD18	SAD19	SAD20	SAD21	SAD22	SAD23	SAG
01111	SAD24	SAD25	SAD26	SAD27	SAD28	SAD29	SAD30	.	SAG
10000	PPE01	PPE02	PPE03	PPE04	PPE05	PPE06	PPE07	PPE08	ACG
10001	PPE11	PPE12	PPE13	PPE14	PPE15	PPE16	PPE17	PPE18	ACG
10010	IFC	.	REN	NRE	.	.	.	.	UCG
10011	.	.	AAU	LPD	.	.	.	.	UCG
10100	DDL0	DDL1	DDL2	DDL3	DDL4	DLL5	DLL6	DLL7	ACG
10101	DDL8	DDL9	DDL10	DDL11	DDL12	DDL13	DDL14	DDL15	ACG
10110	DDL16	DDL17	DDL18	DDL19	DDL20	DDL21	DDL22	DDL23	ACG
10111	DDL24	DDL25	DDL26	DDL27	DDL28	DDL29	DDL30	DDL31	ACG
11000	DDT0	DDT1	DDT2	DDT3	DDT4	DDT5	DDT6	DDT7	ACG
11001	DDT8	DDT9	DDT10	DDT11	DDT12	DDT13	DDT14	DDT15	ACG
11010	DDT16	DDT17	DDT18	DDT19	DDT20	DDT21	DDT22	DDT23	ACG
11011	DDT24	DDT25	DDT26	DDT27	DDT28	DDT29	DDT30	DDT31	ACG
11100	.	.	.	.	.	.	.	.	
11101	.	.	.	.	.	.	.	.	
11110	.	.	.	.	.	.	.	.	
11111	.	.	.	.	.	.	.	.	

CODING CHART (continued)

DDDDD 87654	DDD 321 = 000	001	010	011	100	101	110	111	GROUP
	READY CLASS (101)								
00000	RFC	.	.	.	.	.	.	.	RFC
00001	.	.	.	.	.	.	.	.	RFC
00010	.	.	.	.	.	.	.	.	RFC
00011	.	.	.	.	.	.	.	.	RFC
00100	.	.	.	.	.	.	.	.	
00101	.	.	.	.	.	.	.	.	
00110	.	.	.	.	.	.	.	.	
00111	.	.	.	.	.	.	.	.	
01000	ETO	ETE	NRD	.	.	.	.	.	ARG
01001	.	.	.	.	.	.	.	.	ARG
01010	.	.	.	.	.	.	.	.	ARG
01011	.	.	.	.	.	.	.	.	ARG
01100	SDA	SST	SDI	SAI	TCT	.	.	.	ARG
01101	.	.	.	.	.	.	.	.	ARG
01110	.	.	.	.	.	.	.	.	ARG
01111	.	.	.	.	.	.	.	.	ARG
10000	AAD0	AAD1	AAD2	AAD3	AAD4	AAD5	AAD6	AAD7	AAG
10001	AAD8	AAD9	AAD10	AAD11	AAD12	AAD13	AAD14	AAD15	AAG
10010	AAD16	AAD17	AAD18	AAD19	AAD20	AAD21	AAD22	AAD23	AAG
10011	AAD24	AAD25	AAD26	AAD27	AAD28	AAD29	AAD30	IAA	AAG
10100	AEP0	AEP1	AEP2	AEP3	AEP4	AEP5	AEP6	AEP7	AAG
10101	AEP8	AEP9	AEP10	AEP11	AEP12	AEP13	AEP14	AEP15	AAG
10110	AEP16	AEP17	AEP18	AEP19	AEP20	AEP21	AEP22	AEP23	AAG
10111	AEP24	AEP25	AEP26	AEP27	AEP28	AEP29	AEP30	IEP	AAG
11000	AES0	AES1	AES2	AES3	AES4	AES5	AES6	AES7	AAG
11001	AES8	AES9	AES10	AES11	AES12	AES13	AES14	AES15	AAG
11010	AES16	AES17	AES18	AES19	AES20	AES21	AES22	AES23	AAG
11011	AES24	AES25	AES26	AES27	AES28	AES29	AES30	IES	AAG
11100	AMP0	AMP1	AMP2	AMP3	AMP4	AMP5	AMP6	AMP7	AAG
11101	AMP8	AMP9	AMP10	AMP11	AMP12	AMP13	AMP14	AMP15	AAG
11110	AMP16	AMP17	AMP18	AMP19	AMP20	AMP21	AMP22	AMP23	AAG
11111	AMP24	AMP25	AMP26	AMP27	AMP28	AMP29	AMP30	IMP	AAG

3. ELECTRICAL SPECIFICATIONS

3.1 General

This chapter presents the detailed electrical specifications for PIL. As discussed in chapter 1, the link between devices on the loop is a two-wire electrically balanced line. One of the wires is called the reference line and all voltage measurements are made with respect to the reference.

The voltage waveform at the input terminals of a device is specified and all devices must be able to correctly receive this waveform. Assuming that this input waveform requirement is satisfied for all devices on the loop, all other considerations such as distance between devices, transmitter output waveform, etc. are relatively unimportant.

Indeed, there may be instances when the designer requires certain additional capabilities from the link between devices that are not provided within this specification, such as longer distance or better noise immunity, for example. The designer is free to implement the link in any manner he chooses, provided that, at the receiver input terminals, the resulting waveform is both functionally and electrically indistinguishable from this specification. This may require level translation, data rate conversion, or any of a number of other electrical, functional, or mechanical adaptations.

In the interest of compatibility, however, it is further required that all PIL devices adhere to additional specifications regarding the output terminals of each device, and also that the cable between devices meet certain criteria for use in common systems where the distance between devices is no more than 100 meters. These specifications are also given.

Any link with mechanically compatible PIL connectors, cables, etc. must also be functionally and electrically compatible with these specifications to prevent inadvertant connection to non-standard links. Externally inaccessible links, as between multiple devices within a single mainframe, may be implemented in any manner deemed appropriate.

These specifications are in general based upon an implementation using simple pulse transformers within the device. One transformer provides isolation and level conversion from the input terminals to the PIL interface integrated circuit and a second transformer performs a similar operation for the output terminals. While this specification is not intended to preclude implementations which do not use transformers, it is strongly recommended that transformers be used. It may in fact be quite difficult or even impossible to achieve these specifications with other technology.

FIGURE 3-1: WAVEFORM DEFINITION

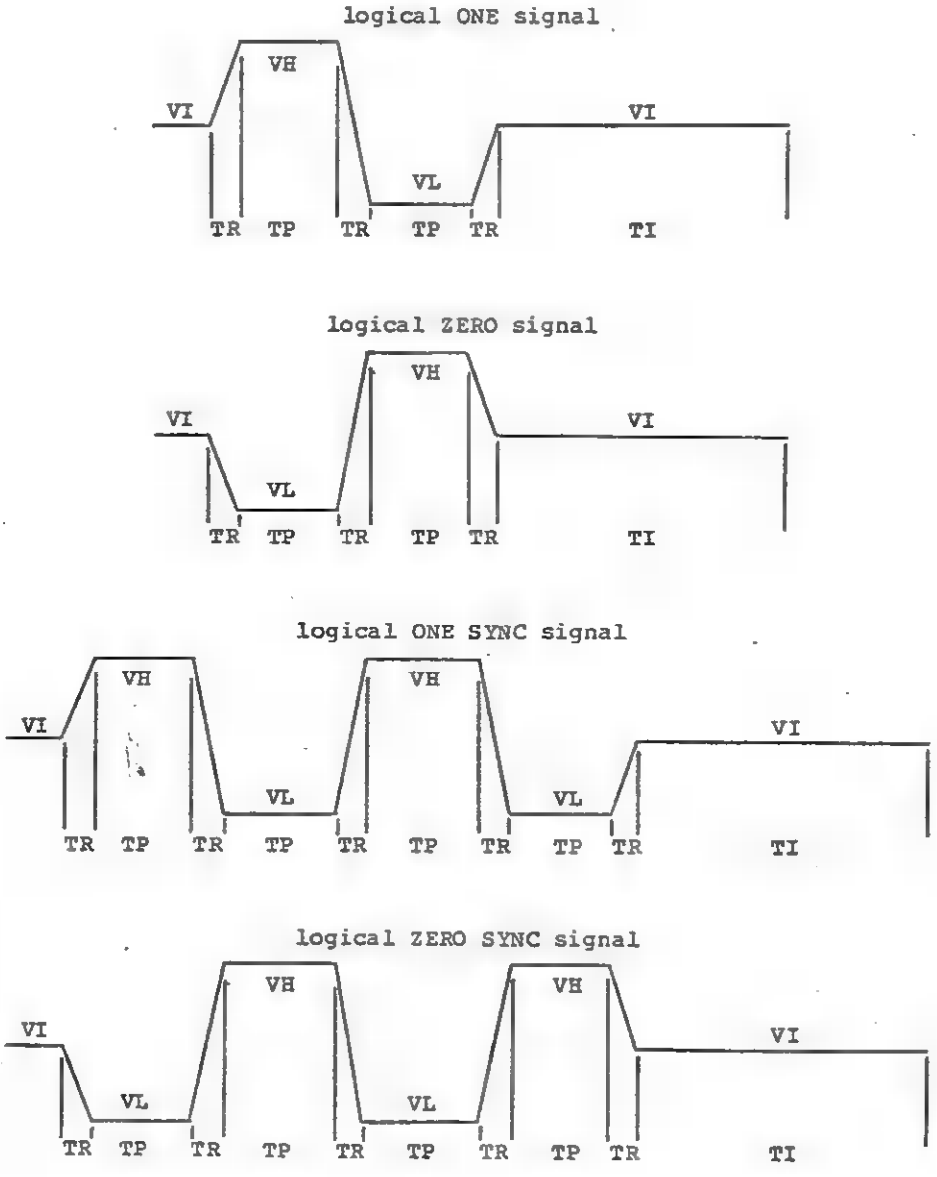
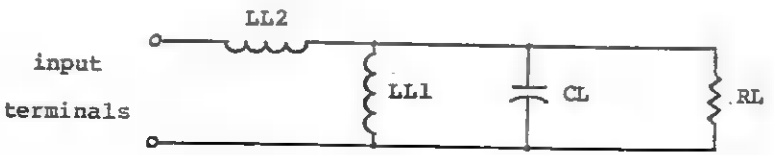


Figure 3-1 defines the meaning of the basic waveforms on the link between devices in terms of logical values. VH, VL, and VI are the voltage values of the high pulse, the low pulse, and the idle time respectively. TR is the rise time from any level to any other level. TP is the pulse width at the high or the low level. TI is the time at the idle level.

As discussed in chapter 1, complete frames are composed of sequential combinations of these basic waveforms. Eleven bits are sent in each frame, the first of which is a sync bit.

3.2 Input Specifications

The input terminals shall present the following equivalent circuit:



parameter	minimum	maximum
RL	750 Ohms	3000 Ohms
CL		400 pF
LL1	1.9 mH	
LL2		4.0 uH

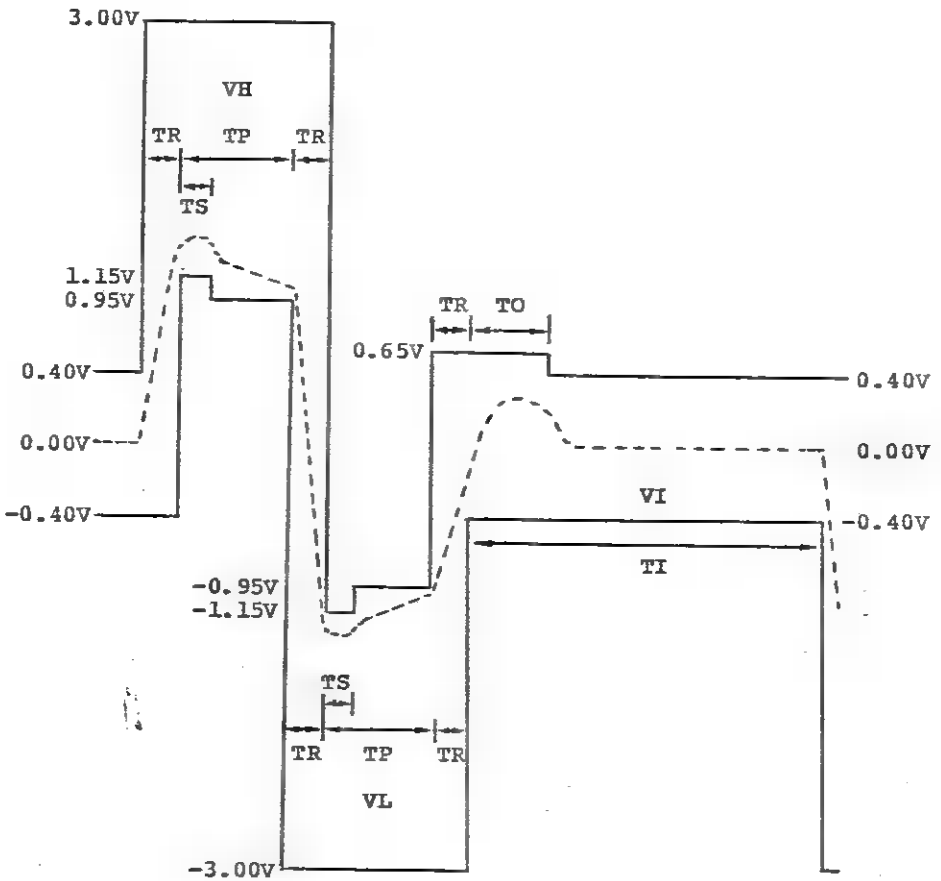
FIGURE 3-2: INPUT EQUIVALENT CIRCUIT

All impedances shall be measured at a frequency of 1 MHz. The input circuit shall not clamp the input signal when the signal is within specified limits.

The input terminals must be isolated from earth (chassis or safety) ground and the device common. The minimum breakdown voltage shall be 500 V. The capacitive coupling from either signal line to ground or common shall be no more than 50 pF within the device. The resistive coupling from either signal line to ground or common shall be greater than 10 MegOhms within the device at 40 degrees C and 80% relative humidity.

To provide immunity to noise and distortion, the use of Schmitt trigger type receivers with hysteresis is required in the input circuitry. The high state (VH) shall be sensed when the

FIGURE 3-3: INPUT WAVEFORM SPECIFICATION



parameter	minimum	maximum
TR	20 nS	300 nS
TP	650 nS	1500 nS
TI	1700 nS	10000 nS
TS	150 nS	
TO		1500 nS

input voltage rises above a threshold value (VTH+) and shall no longer be sensed when the voltage falls below a different threshold value (VTH-). Likewise, the low state (VL) shall be sensed when the voltage falls below a threshold value (VTL-) and shall no longer be sensed when the voltage rises above a different threshold value (VTL+). The idle state (VI) is sensed whenever neither the high nor the low state is being sensed. In both cases, the magnitude of the leading edge threshold must be greater than the magnitude of the trailing edge threshold (hysteresis, HYST). The following table outlines the specifications for the PIL input receivers:

TABLE 3-1: RECEIVER SPECIFICATIONS

parameter	minimum	maximum
VTH+	0.65 V	1.15 V
VTH-	0.40 V	0.95 V
VTL-	-0.65 V	-1.15 V
VTL+	-0.40 V	-0.95 V
HYST	0.09 V	

Furthermore, high or low pulses of 20 nS or less duration must not be sensed. High or low pulses of 650 nS or more duration must be sensed. If the voltage is within the idle range for less than 300 nS, the idle state shall not be sensed. The idle state must be detected when the voltage is within the correct range for 1.7 uS or longer.

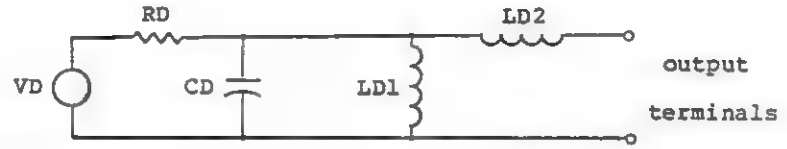
The required input waveform is illustrated and specified in detail in figure 3-3. As long as the waveform remains within the boundaries shown, the data must be correctly received. The diagram shows only the logical ONE signal, however, the logical ZERO signal only requires a sign change on the voltage values. The extension of the diagram to sync signals is also obvious.

In addition to those specifications shown in the diagram, it is also required that the value of TP vary not more than 10% from its mean value within a single frame. Furthermore, the idle time after the last bit of the frame shall be no less than 5.00 uS. Addition indicates that a bit time may be as little as 3.06 uS or as much as 13.90 uS while a sync bit might as short as 4.40 uS or as long as 17.50 uS. Similarly, the minimum time from the beginning of one frame to the beginning of the next could be from 38.30 uS to 156.50 uS. There is no maximum time between frames.

3.3 Output Specifications

The output terminals shall present the following equivalent

circuit:



parameter	minimum	maximum
RD	90 Ohms	105 Ohms
CD		400 pF
LD1	1.9 mH	
LD2		4.0 uH

PIGURE 3-4: OUTPUT EQUIVALENT CIRCUIT

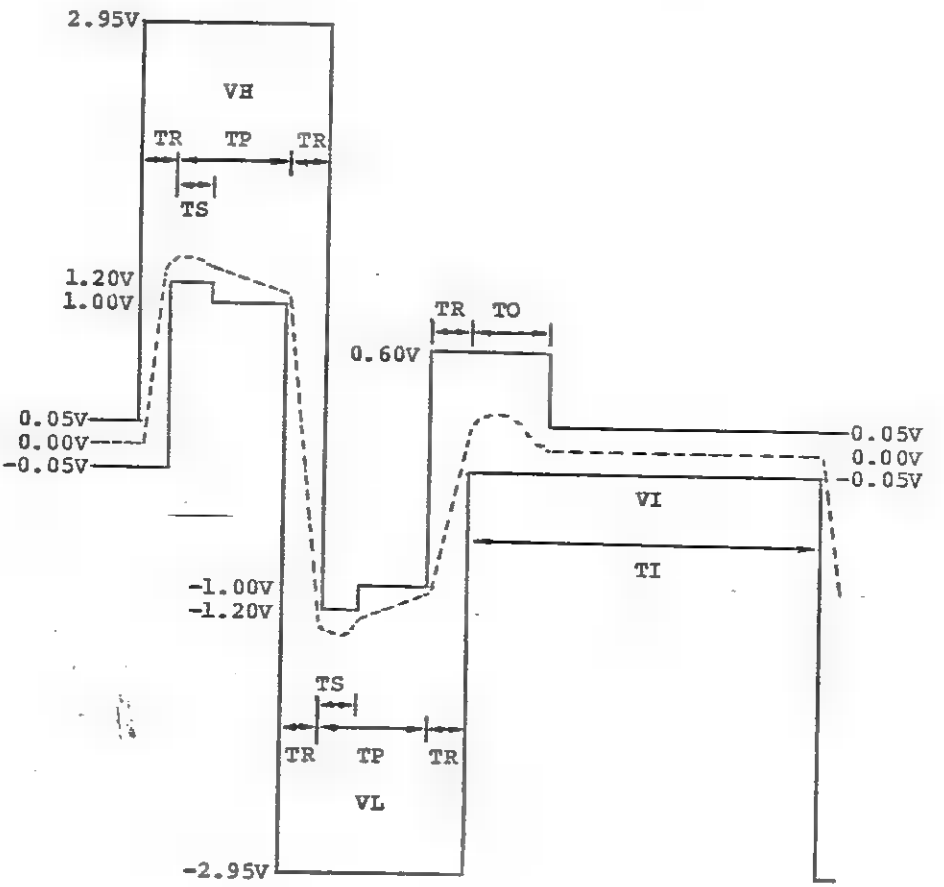
All impedances shall be measured at a frequency of 1 MHz. Isolation requirements for the output circuitry are identical to that required of the input circuitry.

The required output waveform is illustrated and specified in figure 3-5. As long as the waveform remains inside the boundaries shown and standard cable of correct length is used, the waveform will be received correctly by a standard device. In general, these values are similar to the input waveform values with 50 mV and 50 nS of noise margin added. Additionally, the idle time after the last bit of the frame shall be no less than 5.00 uS. These values indicate that the output bit time is from 3.41 uS to 13.30 uS and the sync bit time from 5.05 uS to 16.50 uS. The minimum time from the beginning of one frame to the next is from 42.40 uS to 149.50 uS. There is no maximum time between frames.

3.4 Interface Cable Specifications

The cable length from one device to the next shall be no more than 100 meters. The characteristic impedance of the cable shall be 100 Ohms plus or minus 20% for cables no longer than 10 meters and 100 Ohms plus or minus 10% for longer cables. When the source impedance and the load are both equal to the cable characteristic impedance and a step input is applied, the 0-90% risetime of the cable shall be no more than 200 nS. For distances greater than 10 meters the cable shall be of the shielded twisted-pair type. For floating devices, the shield shall not be connected to device common.

FIGURE 3-5: OUTPUT WAVEFORM SPECIFICATION



parameter	minimum	maximum
TR	20 nS	150 nS
TP	800 nS	1450 nS
TI	1750 nS	9950 nS
TS	200 nS	
TO		1450 nS

3.5 Interference and Susceptibility

The interface shall be in compliance with VDE and FCC EMI requirements for class B devices.

The interface shall also comply with the Radiated Susceptibility Test as specified in section 765.004 of "HP Design Standards - Environmental Tests". For the purposes of this test, ■ device shall have its output connected to its input via ■ 10 meter cable and transmit frames to itself. With the cable in the external electric field and polarized in the direction of the field, there shall be no loss of data or change in data.

4. MECHANICAL SPECIFICATIONS

As mentioned previously, due to the special requirements of PIL ■ special connector set is used. Devices may either use a panel connector or cables affixed directly to the device through strain reliefs (figure 4-1). Cable connectors (figure 4-2) are designed to non-invertible and non-reversible and have a positive detent both in panel connectors and in each other for running cable "splices". In short, they have been designed to be as foolproof as possible.

The connector contacts were chosen for high reliability and long life. For reference, the contacts should be equivalent to those manufactured by ITT Canon, part number 031-9540-000 (male) and part number 030-9542-001 (female). The connector body is of molded polycarbonate while the integral strain relief is of PVC.

For applications where the distance from one device to the next is 10 meters or less, relatively inexpensive "zip" cord cable may be used. The wires are 24 AWG stranded (26 X 38 AWG) individually tinned copper. The wires are spaced 0.060 inches center-to-center with a PVC jacket, 0.065 X 0.130 inches. A polarity rib must be visible on the jacket.

For applications requiring longer distances up to 100 meters, cable which satisfies the more stringent electrical specifications given in the previous chapter must be used, that is, shielded twisted-pair cable whose characteristic impedance is 100 Ohms plus or minus 10%.

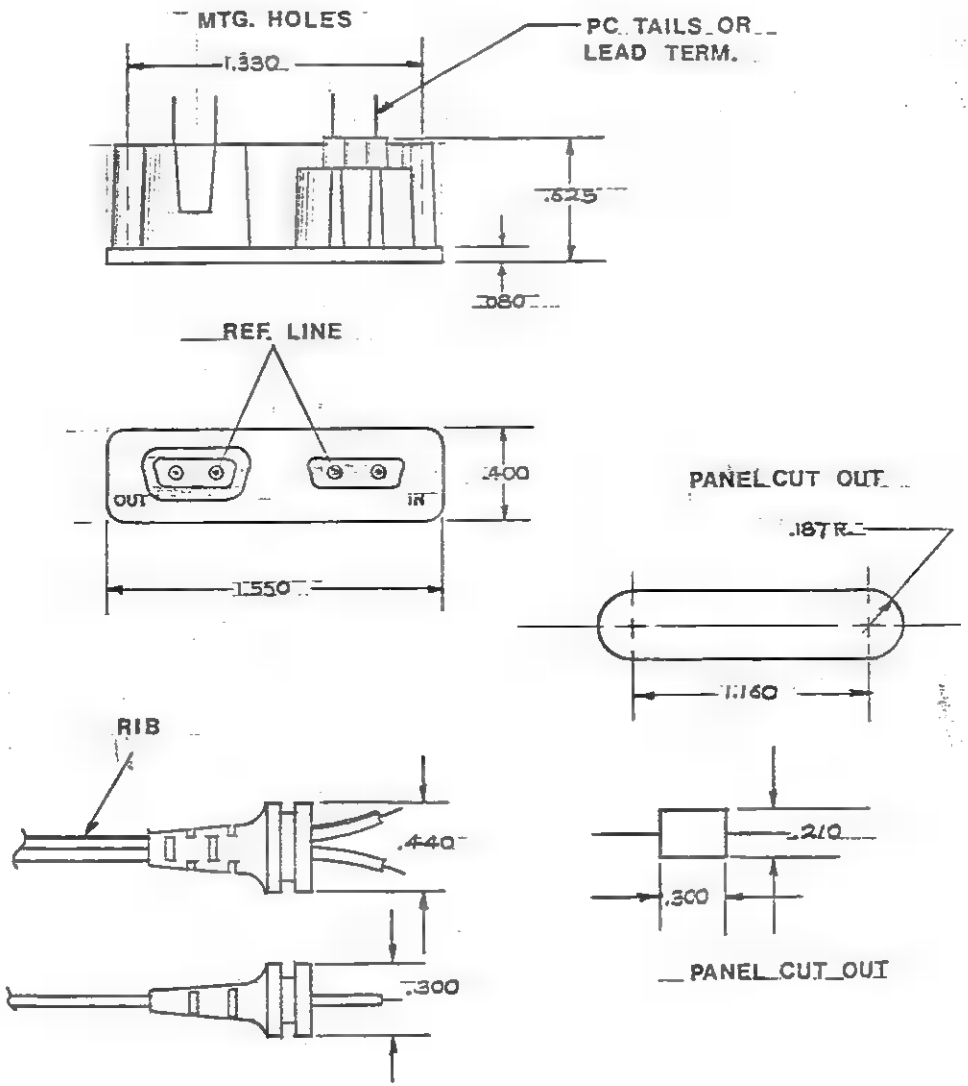


FIGURE 4-1: DEVICE CONNECTORS

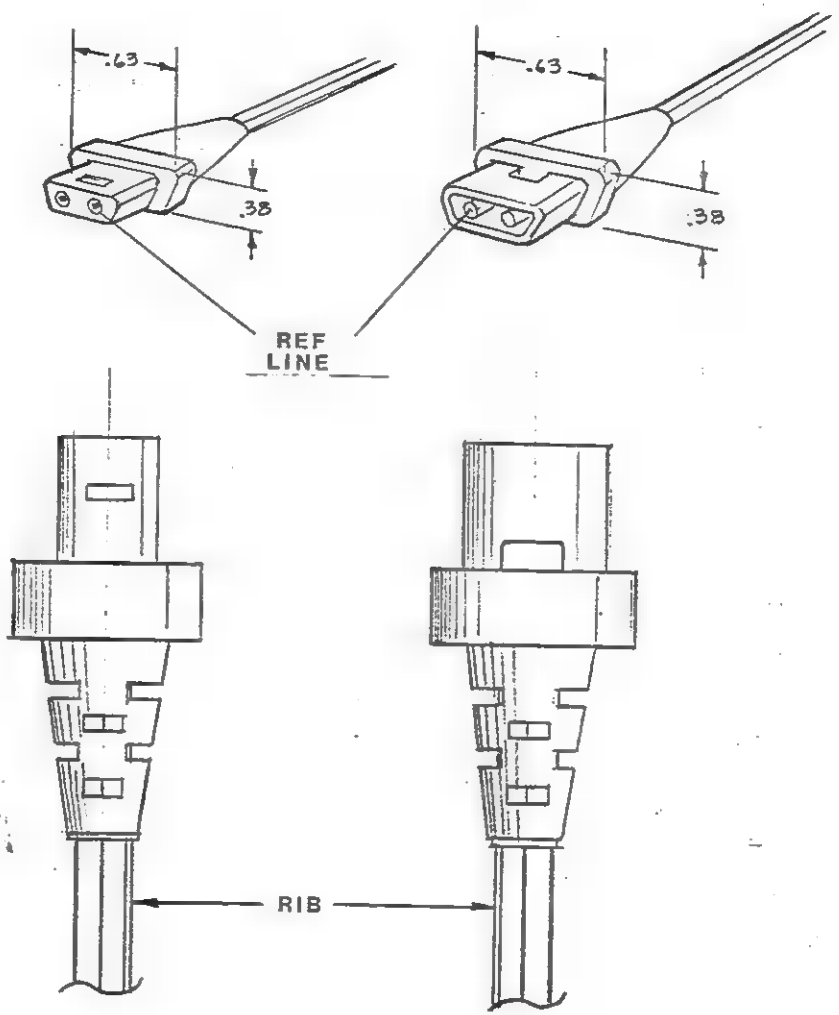


FIGURE 4-2: CABLE CONNECTORS

5. SYSTEM GUIDELINES

5.1 System Compatibility

This interface sytem offers a wide range of capability from which to choose the appropriate interface functions to fit different applications. Within most interface functions a number of options are available. In addition, the designer has freedom to select all the device dependent capabilities contained within the device functions.

It is the responsibility of the designer to define the complete capability of a device (interface system choices and related device dependent interactions) so that the end user of the device can efficiently interface and program the device for appropriate system applications.

Also, devices designed to this interface system may have a wide range of capability relative to their ability to communicate over the interface. This document does not cover the operational characteristics of devices, only the functional, electrical, and mechanical capabilities of the interface system.

The burden of responsibility for system compatibility at the operational level is on the user. The user must be familiar with all device characteristics interacting with the interface system (for example, device dependent program codes, output data format and codes, etc.).

In particular, compatibility will be greatly enhanced if the ASCII code is used by all devices for device dependent messages whenever possible. In addition, it is recommended that device dependent commands be avoided unless absolutely necessary. Simple ASCII programming codes or ASCII escape sequences are preferable for the implementation of simple "friendly" controllers.

5.2 System Configuration

If all devices on the loop use simple addresses only, the maximum number of devices in a PIL system is 31. If all devices use extended addressing, the maximum number is 961.

In general, an interface system shall contain one or more devices containing at least one T function, one L function, and one C function.

If all the T functions include the use of the ton local message and all the L functions include the use of the lon local message, a system may be operated without a C function while the

ton message is true in one and only one of the T functions and the lon message is true in one or more of the L functions. The ton and lon messages are normally provided by local switches. This permits rudimentary, manual system operation.

All system configurations containing more than one controller must satisfy the following conditions: (1) There shall not be more than one C function in a system which is sending the scl local message true at any time. (2) Every controller in the system shall be able to pass and receive control of the interface.

Because of the serial nature of PIL, all devices must be powered on in order for the system to function. If desired, the designer may implement the interface circuitry such that when the device is powered off, power remains on for the interface chip to act as a simple repeater.

5.3 Address Assignment

Normally, a device will be assigned a single talk and a single listen address to perform the essential tasks. It may be useful to design a device with multiple talk (or listen) addresses to facilitate system requirements. A device could be assigned two talk addresses, for example, one to output raw data, the other to output processed data. If this is necessary, it is recommended that two-byte addressing be used for these types of devices. Care should be given to minimize the use of such multiple addresses as later system configurations may be restricted due to excessive use of the simple addressing capability.

While not specifically required, it is strongly recommended that all devices implement one of the auto addressing interface functions so that the controller may assign addresses to all devices in the system. This will greatly enhance compatibility and ease of programming. When these devices are powered on or when they receive the auto address unconfigure command, they may respond to either a preset address or local address switches if desired. It is important to note that the auto address unconfigure command has no effect on the T or L interface functions. If a device is in an addressed state when the AAU message arrives, it will remain in that state and respond accordingly to messages which follow. In general, the messages which follow will immediately assign new addresses. Note that these messages also have no effect on the T and L functions. In short, the act of configuration (or unconfiguration) which is the assignment of addresses to particular devices is distinct and separate from the act of addressing a particular device to talk or to listen. Careful study of the state diagrams may be necessary to completely understand the ramifications of this.

A device that contains a T or TE function may be assigned any value for the least significant five bits of its MTA (my talk address) message code other than 11111. This code, defined as UNT, is provided for the controller to return all devices to the

talker idle state. Two or more T functions shall not be assigned the same value for these bits. A TE function shall not be assigned the same value for these bits as that assigned to a T function. A device that contains both a T and an L function may have the same value for the least significant five bits of both the MTA and MLA (my listen address) messages.

A device that contains an L or LE function may be assigned any value for the least significant five bits of its MLA message code other than 11111. This code, defined as UNL, is provided for the controller to return all devices to the listener idle state. Two or more L functions or LE functions may have the same value for these bits.

A device that contains a TE or LE function may be assigned any value for the least significant five bits of its MSA (my secondary address) message code other than 11111. Two or more TE functions shall not be assigned the same value for both their MTA codes and the same value for both their MSA codes. Two or more LE functions may be assigned the same values for both codes, however. If a device has both a TE and an LE function, it may be assigned the same value for the least significant five bits of both the MTA and MLA codes and both functions may utilize the same MSA code as well.

In general, and particularly when auto addressing or auto extended addressing is used, the T and L function within a specific device will be assigned and will use the same address. Auto multiple addressing is intended specifically for those cases where there are multiple devices or addressable device functions within one mainframe. A block of secondary addresses is reserved by the device and these are assigned to the various TE and LE functions as the designer may choose.

While it is possible to mix devices which use simple addressing, extended addressing, and multiple addressing on the same loop, this should only be done with a good deal of caution and thought for the interactions of the various addresses. The controller will need to be quite sophisticated to handle the configuration task.

5.4 Asynchronous Loop Operation

As mentioned in chapter 1, the normal mode of operation for the loop is such that only one message frame is in transit around the loop at any given time. DOE and IDY messages have a bit reserved to carry a device's request for service back to the active controller. IDY messages have several bits reserved as well which can be assigned to individual devices for service request purposes. These methods of notifying the controller of a need for service on the loop are perfectly satisfactory for the large majority of applications. These methods require a committment on the part of the controller, however, to periodically poll the loop for service requests via the IDY message whenever device dependent transmissions are not taking place. In a few rare instances the system designer may determine

that it is necessary to allow the loop to be totally quiescent and yet still have the controller notified of a need for service. Also, there may be applications where rapid response to service requests is necessary and the interval between frames of slow device dependent message transmission is much too long. In these situations, asynchronous loop operation becomes necessary and multiple frames may be on the loop at one time.

The implementation of a PIL system which uses asynchronous operation should be approached with utmost caution. The serial nature of the loop and the fact that device clocks are not synchronized means that multiple frames may "stack up" at one device and some data may be lost. In addition, implementation of the first PIL integrated circuits was not totally in agreement with the functional specifications of this document (which ocurred later), and this minor difference can, under certain conditions, cause a similar loss of data. These comments apply only to asynchronous operation of the loop and do not have any effect otherwise. For this reason, it is important to think of asynchronous loop operation as a separate, non-standard mode of operation in which only certain, carefully defined tasks are accomplished. The burden of handling the possibilities mentioned here rests with the active controller and only the active controller enables and disables the asynchronous mode of operation. This may affect the controller designer, the system programmer, or both.

The device which is designated to be the system controller is the only device which can send the IFC message. It may do this at any time it chooses. In the general case, this is an asynchronous operation and the system controller should follow the same sequence every time, even at initial system power up. The system controller should send the IFC message and then wait for it to return. If it does not return within some relatively long time period, say, one second, the system controller should send the IFC again and wait, repeating this sequence until the IFC returns. It is possible that other frames will be received by the system controller before the IFC returns. These frames must not be retransmitted; they must be destroyed by the system controller to prevent a condition in which a frame's destination device is no longer active permitting the frame to endlessly circulate around the loop. When the IFC returns, the system controller should then source the RFC message to complete the handshake. While unlikely, it is barely possible that other extraneous frames could be received by the system controller after the IFC has returned. These must also be destroyed. The RFC will then return (it should only be sent once) and the interface functions are now initialized and the loop is flushed of any extraneous frames. The IFC sequence is not intended to preserve message frames on the loop so the system controller need not concern itself with the destroyed frames. Any system controller must be able to execute this sequence, regardless whether it possesses asynchronous IDY capability or not.

For the situation where a controller needs to respond rapidly to service requests during a very slow device dependent transmission, the active controller is permitted to source the IDY message at any time (provided the controller is implemented

with the proper capability subset). The IDY is immmediately retransmitted by all devices and will return to the controller relatively rapidly with service request and, optionally, parallel poll information.

The IDY will very likely overtake and pass whatever other message is presently on the loop. The functional specifications require that this other message not be affected in any way by the passage of the IDY. Loop operation should continue without any loss of data. Provided that the controller only source one asynchronous IDY at a time, there will be no problems with frames "stacking up" and being lost. When the IDY returns to the controller, it is then free to source another IDY without fear of running into this kind of problem.

However, the active controller is not constrained to sourcing the asynchronous IDY only during device dependent message transmission, but may source this message at any time. The IDY could be sent after a CMD, or after the RFC following a CMD, or after some other RDY message which the controller itself has sourced. In the first PIL integrated circuits (marked 1LB3-) a difficulty exists if the controller sources an asynchronous IDY after an RFC message has been sent. If the IDY overtakes the RFC in a device which uses this chip, the RFC is destroyed. In order to preserve compatibility, therefore, all controllers which have asynchronous capability must either (1) never source an asynchronous IDY after an RFC or (2) implement measures to recover from the possible destruction of the RFC by the IDY. The second option might involve sending the RFC over again to guarantee a complete handshake but this then raises the possibility that there are now two RFC messages on the loop if the first was not in fact destroyed. Some other RDY frame (NRD perhaps) might be used to flush the loop in this instance. In this case, the controller must be able to properly respond whether there are zero, one, or two RFC messages on the loop. No other messages are destroyed by the asynchronous IDY from the controller, provided that only one IDY is on the loop at a time, as mentioned above.

In the case where the loop is not polled for service requests, the controller may send the EAR command (enable asynchronous requests), the RFC, and then simply stop sending any messages. When devices receive this command, they enter a mode in which, if they have the capability and require service, they may source their own asynchronous IDY. This message will not be retransmitted by the active controller, but will cause it to resume normal operation which includes handling the service request. Because it is remotely possible that in this mode a number of devices could generate the IDY at about the same time, frames could "stack up" and be destroyed. Therefore, it is required that normal loop operation be suspended while in this mode. The controller may only source CMD messages while the asynchronous request mode is in effect, and all CMD messages (except EAR and LPD) disable this mode and return the loop to normal operation. Note that parallel poll information contained in these IDY messages from other devices is not accurate since they have not, in general, travelled completely around the loop.

The previous discussion regarding RFC messages is equally applicable in this mode. The controller should not require the RFC following the EAR to return as it could be overtaken or even destroyed by an asynchronous IDY from one of the devices. The completion of the handshake is relatively unimportant in this instance since the controller will immediately send other commands to service the request, disabling the mode anyhow. But the controller must still be able to handle this situation. The controller should handle the return of the loop to normal operating mode in the same manner as the system controller handles the IFC sequence. The CMD is sent first which disables asynchronous request mode. NUL is recommended as it does not otherwise modify the state of the interface. Other IDY messages may return before the CMD. They should be ignored (not retransmitted). If the CMD does not return after a period of time, it should be sent again, and so on, until it does return (there is a slight possibility that the CMD could be destroyed by frame "stack up"). The RFC can then be sent. One or more additional IDY message may possibly arrive before the RFC returns. They should be ignored as were the earlier ones. When the RFC returns, the loop is flushed and has been returned to its normal, synchronous operating state. The controller can now continue to service the request. Very much the same sequence is used to bring up a loop which has been powered down with the LPD command and left in the asynchronous request mode so that a device other than the active controller may initiate the loop wake-up.

It is easily seen that the implementation and programming of controllers with asynchronous capabilities is no mean task. It should only be attempted by designers and programmers who have a complete and total understanding of loop protocol and the internal details of the PIL integrated circuit chip operation.

5.5 Operational Sequences

The following collection of sequences is not intended to be exhaustive, nor is it necessarily true that the sequences here represent the only way to accomplish the given task. They are included merely as examples of one possible method to perform the particular operation and as further help in understanding the protocol used for PIL.

Initial System Power On

IFC The system controller initially sends the interface clear
. command at regular, slow intervals until it returns,
. indicating that now all devices on the loop have been
. powered up, are properly receiving frames, and are
IFC initializing their interface functions.

RFC Now the system controller sends the ready for command message so all devices can indicate tney have finished their initialization and are ready to receive the controller's

next command.

. The system controller now procedes to carry out normal
. operations on the loop.
.

Talk-only, Listen-only System Power On

DAB1 The talker immediately sends its first data byte and
. continues to send it at slow, regular intervals until
. it returns, indicating that the other devices on the
. loop are powered up and are properly retransmitting or
DAB1 receiving frames.

DAB2 The talker now sends the rest of its data string at
DAB3 normal speed.
.
.
DABn

DAB1 When the transmission is finished, the talker immediately
DAB2 begins sending its next message string. No EOT message
. is sent in a talk-only, listen-only system. Transmission
. continues in this manner until the ton local message
DABn is switched off.

Data Transfer

UNL If necessary, the controller will first inhibit any
RFC previous listeners with the unlisten command.

LAD1 The controller now sends the appropriate listen address
RFC command to enable the listener. More than one listener
LAD2 can be enabled if desired.
RFC
.
.

TAD The talk address command enables one device to send
RFC data when it receives the proper ready frame.

SDA The controller now sends the send data ready frame
DAB1 which the talker replaces on the loop with its first
data byte.

DAB2 The talker continues to send the remaining data bytes
DAB3 of its message.
.
.
DABn

ETO After the end of its message the talker sources the
. end of transmission message. The ETO indicates that
. all frames returned and error checked correctly. The

. controller replaces the ETO with its next loop
. operation.

Controller Interruption of Data Transmission

DAB5 Assume the talker has sent the fifth data byte.

DAB6 The controller decides that it has more important tasks
NRD to perform and holds the talker's sixth data byte, and replaces it with the not ready for data message. The NRD goes through the talker signalling it that it must immediately end its transmission and returns to the controller.

DAB6 The controller now retransmits the held up data byte.

ETO When the data byte returns to the talker, it sends the
. end of transmission message. The ETO is replaced on
. the loop by the controller's next operation. If the
. controller now transmits the send data frame again
. (without having unaddressed the talker), the data
. transmission will continue where it left off.

Serial Poll

IDY The controller sends the identify message and receives it back with the service request bit set indicating that one or more devices need service.

UNL The unlisten command prevents other devices from
RFC receiving the various devices' status bytes.

TAD1 The first device is addressed to talk.
RFC

SST The controller sends the send status ready frame which
DAB the talker replaces on the loop with its status byte or bytes.

ETO The talker sends the end of transmission message and
TAD2 the controller replaces it with the next device's
RFC talk address.

SST This same sequence is repeated for each device on the
DAB loop. The controller listens to the status bytes via
ETO its local listen capability and determines which loop
. devices have a need for service and which do not. If
. the SST message simply returns to the controller
. instead of a data byte, it knows that the particular
. device does not implement the serial poll function
. (and hence cannot need service). It can continue to
. poll the other devices until it is finished and/or
. handle the service requests as desired.

Parallel Poll Configuration and Operation

UNL
RFC
PPU
RFC — The controller may choose to send the unlisten command to prevent unwanted devices from reacting to parallel poll enable commands and/or the parallel poll unconfigure command to reset any previous parallel poll configuration on all devices.

LADn
RFC — The controller will now send the listen address of the device to be configured for parallel poll.

PPE13
RFC — The parallel poll command enables the addressed device(s) to respond, in this case, by ORing a one onto bit 3 of an IDY message.

UNL
RFC — The unlisten command is sent so that the previous device will not be changed by any succeeding PPE messages. This same sequence (LAD, PPE, UNL) is repeated for each unit to be configured for parallel poll. Individual devices can be disabled (rather than all devices together, as with the PPU command) by substituting the PPD command for the PPE command in this sequence.

IDY — The controller executes a parallel poll operation by sending the IDY message. The message will return with its data bits set according to the configuration commands and the devices' individual status bits. One device can be assigned to each bit for rapid response to at most eight devices.

Assign Auto Addresses

AAU
RFC — If necessary, the controller will send the auto address unconfigure command to reset any previous addresses and make all devices which implement this function ready to receive new addresses.

AAD1
AAD2
AAD3
AADn — The controller sends the first auto address ready frame and the first device on the loop accepts it as its new address. It increments the address bits and then sends the frame on to the next device, which, in turn, takes the new value as its address and increments the frame again, sending it on to the next device. This process continues around the loop until the modified frame returns to the controller, indicating the number of devices on the loop. If the frame returned with address 31, there may be too many devices on the loop. To determine this, the controller could now send AAD1 again. If it returns unchanged, there are exactly the maximum number of devices. If it has been incremented, there are too many devices; the controller should inform the user of an error. The loop will not work.

The controller has now configured the loop with simple addresses and proceeds to perform its tasks. Devices

. will no longer respond to other AAD frames until the
. AAU command is sent once again.

Control Passing

TADn RFC — The currently active controller addresses the device to which it will pass control of the loop as a talker.

TCT — The controller then sends the take control ready frame
. which the new controller replaces on the loop with its
. first operation. If, instead, the TCT returns to the
. previous controller, it knows that the addressed
. device could not or would not take control. The
. previous controller then resumes control of the loop.

A. CAPABILITY SUBSETS

R (Receiver) Interface Function

All devices must have the complete R function implemented.

AH (Acceptor Handshake) Interface Function

All devices must have the complete AH function implemented.

SH (Source Handshake) Interface Function

Identification		Requirements
SH0	No capability	Omit all states
SH1	Complete capability	Implement all states

D (Driver) Interface Function

All devices must have the complete D function implemented.

L (Listener), LE (Extended L) Interface Functions

Identification		Requirements
L0, LE0	No capability	Omit all states
L1, LE1	Basic capability	Implement all states
2	2 Listen-only mode	Additional to L1, LE1; local message lon not always false
3	3 Unaddress if talk addressed	Additional to L1, LE1; implement optional term; needs talker capability

Only one of L, LE need be implemented in any particular device.

T (Talker), TE (Extended T) Interface Functions

Identification		Requirements
T0, TE0	No capability	Omit all states
T1, TE1	Send data	Implement TIDS, TADS, TACS
T2, TE2	Send status	Implement TIDS, TADS, SPAS

T3, TE3	Send device ID	Implement TIDS, TADS, DIAS
T4, TE4	Send accessory ID	Implement TIDS, TADS, AIAS
5, 5	Talk-only mode	Additional to T1, TE1; local message ton not always false
6, 6	Unaddress if listen addressed	Additional to T1-4, TE1-4; implement optional term; needs listener capability

Only one of T, TE need be implemented in any particular device. Talker capability consists of one or more of T1-4 or TE1-4.

C (Controller) Interface Function

Identification		Requirements
C0	No capability	Omit all states
C1	Basic capability	Implement CIDS, CACS, CSBS, CEIS, CEMS
2	System controller	Additional to C1; local messages sic, scl not always false
3	Respond to service requests	Additional to C1; add CSNS, CSRS (needs listener capability to interpret status bytes)
4	Pass, receive control	Additional to C1; add CTRS and optional term; needs talker capability
5	Parallel poll	Additional to C1
6	Interpret, source, and enable asynchronous IDY messages	Additional to C1
7	Assign auto addresses	Additional to C1

DC (Device Clear) Interface Function

Identification		Requirements
DC0	No capability	Omit all states
DC1	Respond to universal device clear command	Implement all states; omit optional term
DC2	Respond to universal and addressed device clear commands	Implement all states; include optional term

DT (Device Trigger) Interface Function

Identification		Requirements
DT0	No capability	Omit all states
DT1	Complete capability	Implement all states; needs listener capability

PP (Parallel Poll) Interface Function

Identification		Requirements
PP0	No capability	Omit all states
PP1	Complete capability	Implement all states; needs listener capability

SR (Service Request) Interface Function

Identification		Requirements
SR0	No capability	Omit all states
SR1	Basic capability	Implement SRIS, SRSS, SRHS; needs T2 or TE2 capability
SR2	Basic and asynchronous request capability	Implement all states; needs T2 or TE2 capability

AA (Auto Address) Interface Function

Identification		Requirements
AA0	No capability	Omit all states
AA1	Complete capability	Implement all states

AE (Auto Extended Address) Interface Function

Identification		Requirements
AE0	No capability	Omit all states
AE1	Complete capability	Implement all states

AM (Auto Multiple Address) Interface Function

Identification		Requirements
AM0	No capability	Omit all states
AM1	Complete capability	Implement all states

RL (Remote Local) Interface Function

Identification		Requirements
RL0	No capability	Omit all states
RL1	Basic capability	Implement RIDS, RACS, LOCS, REMS; needs listener capability
RL2	Basic capability with local lockout	Implement all states; needs listener capability

PD (Power Down) Interface Function

Identification		Requirements
PD0	Basic capability	Implement POFS, PONS, PUPS; pseudomessage edge always false
PD1	Complete capability; responds to power down command	Implement all states

All devices must implement one of PD0, PD1.

DD (Device Dependent Commands) Interface Functions

Identification		Requirements
DD0	No capability	Omit all states
DD1	Responds to one or more device dependent listener or talker commands	Implement all states; need listener capability for DDL commands, talker capability for DDT commands

B. MESSAGE GLOSSARY

Local Messages and Pseudomessages from Device Functions to Interface Functions

arq - Asynchronous request. To SR (service request) function. The device uses arq to signal the function to send the asynchronous IDY (identify) message back to the controller to request service. This will only happen if the device has this capability and has been enabled to do this by the EAR (enable asynchronous requests) command. The arq message must go false as soon as the function enters the state which sources the IDY (ARSS). It may go true again upon return to the standby state (SRSS).

edge - Wake-up signal. Pseudomessage to PD (power down) function. If the device has the full power down capability and has been powered down with the LPD (loop power down) command, the function will be in the power off state, even though the power switch will still be on. In this condition, any frame (actually, any pulse) on the loop will generate the edge message and cause the function to bring the device back to full power. The edge message is a pulse which must only last long enough for the device to completely power on to its initial state. When edge goes false, the device will then be ready for normal operation.

fon - Force on. To PD (power down) function. Under certain conditions, devices which have the full power down capability may need to remain powered up even though they have received the LPD (loop power down) command. The fon local message causes them to go back to PUPS (power up state) rather than go to POFS (power off state). The active controller would use this feature to remain awake, while presently inactive controllers could allow themselves to power down normally. If a device was enabled to send asynchronous IDY messages to wake up the loop, it might also use this message to remain awake.

frtc - Frame transmission complete. Pseudomessage to D (driver) function. This message is generated by the encoder circuitry to signal the D function that it is done transmitting an entire frame and that the function can now return to the idle state to wait to begin transmitting another frame. This message should only be true until the function returns to the idle state.

gta - Go to active. To C (controller) function. When the controller enables a device dependent transmission, it goes to the standby state to wait for completion of the

transmission. If it needs to interrupt the transmission for any reason, it uses this message to force the C function back to the active state before normal completion. The active controller might do this just before sending the NRD (not ready for data) message, the system controller might use it before sending an asynchronous IFC (interface clear) or the active controller might do this to send an asynchronous IDY (identify) message. The gta message should only be true until the function returns to the active state.

gts - Go to standby. To C (controller) function. After the controller has used the gta message to force the C function active to source a message during a device dependent transmission, it uses the gts message to return to the standby state to wait for the normal completion of the device dependent transmission. This message should only be true until the function returns to the standby state.

lab - Local abort. To SH (source handshake) function. This message is used by the active controller. If it is sending a sequence of commands and suddenly finds that it needs to asynchronously source an IDY (identify) or IFC (interface clear, system controller only) message, it uses the lab message to force the SH function back to the generate state. It would otherwise be "hung" waiting for the previous message to return before it could generate the new message. The lab message is only true until the function returns to the generate state.

lon - Listen only. To L (listener) or LE (extended L) function. This message is usually controlled by a manual switch and serves to make the device an active listener even though it has not been addressed. This is only allowed in a special system configuration which has no controller, a talk-only, listen-only system.

ltn - Local listen. To L (listener) or LE (extended L) function. This message is used by the active controller to make itself an active listener without sending out its own listen address, though it could do this if it wished to, also. This message might be used to allow the controller to monitor a device dependent message transfer between other devices (to interrupt it with the not ready for data message, perhaps) or it might also be used if the controller itself was the destination of the data. The ltn message will only be true until the function enters the active state.

lun - Local unlisten. To L (listener) or LE (extended L) function. This message is used by the active controller to return the listener function to the idle state after it had been forced active by the ltn local message. This message will only be true until the function returns to the idle state.

nba - New byte available. To SH (source handshake) function.

When nba goes true, the device is telling the SH function that it has generated another byte for transmission and is ready to have it sent. The nba message must go false before the function can then return to the generate state to handshake the next byte.

pof - Power off. To PD (power down) function. This message is usually sent true by the "off" position of the device power switch. It causes an immediate transition from any state to POFS (power off state). This, in turn, causes all other functions to go to their power off states. This message remains true until the switch is moved to the "on" position.

pon - Power on. To PD (power down) function. This message is a short pulse generated when the device power switch is moved to the "on" position. It causes the PD function to move from the power off state to PONS (power on state). The function remains in PONS until pon goes false. The pon message must last long enough to allow all device and interface functions to go from powered down to powered up and ready for normal operation. When pon goes false, the function enters PUPS (power up state).

rdy - Ready. To AH (acceptor handshake) function. With this message the device indicates to the AH function that it is now ready to receive the next byte of the incoming message string. The rdy message allows transition to ACDS (acceptor data state) which transfers incoming frames to all the other interface functions and device functions. As long as the device is busy (it is interpreting the present message or its buffer is full) the rdy message should be sent false. The device indicates acceptance of the message from the function by setting rdy false. This message handshakes bytes into the device.

rsv - Request service. To SR (service request) function. The device indicates to the SR function a need for service with this message. It causes the function to go to the SRSS (service request standby state). The function then will send the SRQ (service request) message when it has the opportunity, by setting the appropriate bit in data, end, and identify frames, or by sending its own identify frame if enabled to do so. The rsv message is the same as bit D7 of the device's status byte.

rtl - Return to local. To RL (remote local) function. This message is usually generated by a button on the device which returns control of the device functions to manual controls on the instrument itself. The function has the optional capability to ignore this message with the LLO (local lockout) command. The message must not be always true; it will usually be a short pulse.

scl - System controller. To C (controller) function. This message indicates to the function that it is the system controller, the only device permitted to send the IFC (interface clear) message and take control of the loop at

any time. In general, this message will remain true in one and only one device throughout the operation of the interface.

sic - Send interface clear. To C (controller) function. The system controller uses this message to cause the function to go from the idle state to the active state so that it can send the IFC (interface clear) message and take control of the loop. The sic message must be used at initial power on as well as any other time the system controller needs to asynchronously take control of the interface. The message should only be true until the active state is entered.

sync - Valid sync bit received. To R (receiver) function. This message is generated by the decoder circuitry when it recognizes the beginning of a message frame coming in on the loop. It causes the function to go from idle to RSYS (receiver sync state) where it will decide whether the message is for this device or not, and then take appropriate action. The sync message should only be true until the function enters RSYS.

tlk - Local talk. To T (talker) or TE (extended T) function. With this message, the active controller can make itself the talker and source device dependent messages without sending its own talk address, though it could do this as well, if desired. The tlk message should only be true until TACS (talker active state) becomes true. The controller will use this message when it wants to send data (not commands or ready frames) to another device on the loop.

ton - Talk only. To T (talker) or TE (extended T) function. This message is normally generated via a manual switch and causes the device to become the active talker even though it has not received its talk address. This is only permitted in a special system configuration which lacks a controller and is called a talk-only, listen-only system. The function will remain in TACS (talker active state) sending continuous data messages as long as the switch is in this position.

Remote Messages Between the Device and Interface Functions and the Interface Loop

AADn - Auto address n. Auto address group, ready class, 101 100AAAAA. The controller sends AADn to assign simple addresses to devices on the loop. AAAAA represents a five bit binary coded address number n which can range from 0 to 30 (31 is an illegal address, devices will not respond to this value). Each device accepts the incoming value n as its address, increments this value, and sends the modified AAD to the next device on the loop, which, in

turn, does the same. Once a device has received the AAD, it will no longer respond to any other AAD until after it receives the AAU command or is powered off, then on again. The controller uses the address value which returns after going through each device around the loop to determine the number of devices, or to determine if there are too many devices.

AAG - Auto address group. Ready class, 101 100XXXXX. This mnemonic indicates the entire group of auto address ready frames, including simple address, extended and multiple address, and secondary address assignment frames. The controller uses these to assign addresses to devices on the loop in various ways.

AAU - Auto address unconfigure. Universal command group, command class, 100 10011010. The controller uses this command to cause all devices to reset their address assignments prior to being assigned a new set of addresses. After an AAU, devices may respond to address switches, a preset power on address, or no address at all. Except at initial power on, the controller must send AAU before assigning new addresses, else the devices will not respond. IFC (interface clear) does not affect address assignments in any way.

ACG - Addressed command group. Command class, 100 X000XXXX or 100 101XXXXX or 100 110XXXXX. This mnemonic indicates that group of commands to which a device does not respond unless it is addressed as a talker or a listener, depending on the particular command. This group also includes the device dependent commands, DDLn and DDTn, as well as others.

AEPn - Auto extended primary n. Auto address group, ready class, 101 101AAAAA. After the controller has assigned secondary addresses with the AESn frame to a group of devices which can accept extended addresses, it uses the AEPn frame to assign the same primary address to each device in the group. Devices do not modify the frame, they merely accept the address assignment and send the frame to the next device. Other devices do not respond to this frame. After the AEP, the device is configured and can respond to its assigned secondary and primary addresses. AAU will reset the address assignment and ready the device to receive a new address. AAAAA represents the five bit binary address n, which can range from 0 to 30 (31 is an illegal address, devices will not respond to 31).

AESn - Auto extended secondary n. Auto address group, ready class, 101 110AAAAA. AES is used by the controller to assign secondary addresses to extended addressable devices. AAAAA is the binary address n, which can range from 0 to 30 (31 is illegal). Each device accepts the value n as its secondary address, increments this value, and sends the modified frame on to the next device. When the value reaches 31, no other devices respond and the frame simply returns to the controller. The controller

can then use AEPn to assign the primary address to this group of devices. Once configured, the devices no longer respond to the AESn, so the controller can now send it out again to assign extended addresses to the next group of devices on the loop. The primary address for each group must, of course, be unique.

AMPn - Auto multiple primary. Auto address group, ready class, 101 111AAAAA. AMP assigns primary addresses to all devices which use multiple addressing on the loop. AAAAA represents the five bit binary address n, which can range from 0 to 30 (31 is illegal). The controller sends the AMP message and each succeeding device accepts the incoming value as its new address, increments n, and sends the frame to the next device, which, in turn, does the same. The value which returns to the controller indicates the number of multiple address devices on the loop. Following this, the controller sends the ZES command to each device so that it can reserve the proper sized block of secondary addresses. The device is then configured and can respond to its assigned addresses normally. AAU is necessary before devices will respond to new address assignments.

ARG - Addressed ready group. Ready class, 101 01XXXXX. This group of frames is ignored (simply retransmitted) by devices which are not presently addressed. Only active talkers, listeners, and controllers respond to these messages. Presently included are the start of transmission, end of transmission, and not ready for data subgroups and frames. With the exception of NRD, these frames do not travel all the way around the loop back to their sourcing device. In general, they serve a handshake function and the destination device replaces them with another message frame. At present, listeners do not respond to these frames, but may in the future with expansions to loop definition.

* CMD - Command. 100 XXXXXXXX. Commands are one of the major classes of loop frames. They control the operation of the interface functions of each device in a major way, and to a lesser extent, the device functions also. The active controller is the only device which sources command frames (except for asynchronous IFC by the system controller). Every command must be immediately followed by the RFC message to provide devices the oportunity to handshake, that is, to indicate the are ready to receive the next command. Commands are immediately retransmitted by all devices to minimize delay but a copy of the frame is saved by each device to begin execution of the command.

DAB - Data byte. Data or end class, 00X XXXXXXXX. Data bytes are the basic unit of the device dependent message transmission. This is the data which the interface system is designed to handle. The other messages are largely overhead for control purposes. These messages between devices may be coded in any way but it is strongly recommended that ASCII be used wherever possible for

compatibility reasons. The data byte also contains the SRQ bit, C1, which devices may set to indicate to the controller a need for service.

DCL - Device clear. Universal command group, command class, 100 00010100. DCL is sent by the controller to cause all devices which recognize this command to set their device functions to a preset state, whether they are addressed or not. This command does not affect the interface functions. The preset state is defined by the device designer.

* DDLn - Device dependent listener command n. Addressed command group, command class, 100 101XXXXX. A device must be addressed as a listener in order to respond to any one of the 32 possible DDL commands. The particular effect of the specific command is designer determined.

* DDTn - Device dependent talker command n. Addressed command group, command class, 100 110XXXXX. A device must be addressed as a talker in order to respond to any one of the 32 possible DDT commands. The particular effect of the specific command is designer determined.

DOE - Data or end. 0XX XXXXXXXX. This major frame classification includes all the device dependent messages for the interface system. This is the data which is communicated from one device to another and for which the system was designed. DOE frames include the END bit, C2, to indicate an end-of-record condition without terminating the transmission, and the SRQ bit, C1, for devices to indicate a need for service to the active controller. These frames are, in general, sourced by the active talker. Their destination is the active listener(s) on the loop.

EAR - Enable asynchronous requests. Universal command group, command class, 100 00011000. This command is used by the active controller to put all devices which have the capability in a mode where they can source their own IDY message to indicate a need for service to the controller. Normally, the controller is the only device to source IDY frames. After the controller sends the EAR - RFC sequence, it will allow the loop to go idle, until such time as an asynchronous IDY from one of the devices arrives or until the controller must perform some other operation. The controller sends out a command which disables the asynchronous request mode and then resumes normal operation. All commands except EAR and LPD disable the mode, however it is recommended that controllers use the NUL command as it has no affect on the other interface functions. LPD does not disable the mode so that it is possible to have some device other than the active controller wake up a powered down loop.

END - End data byte. Data or end class, 01X XXXXXXXX. The end frame is the same as a data byte except that bit C2 is set to indicate the end-of-record condition to the listener.

This has no affect on the interface functions and the end byte is treated exactly the same as any other data byte as far as the system is concerned. Most ASCII transmissions will indicate end-of-line with a CR, LF pair, for example, but binary data will probably need to use the END message for this function. This does not terminate the transmission.

EOT - End of transmmission. Addressed ready group, ready class, 101 0100000X. This is a subgroup including two messages, end of transmission (error) and end of transmission (OK). The active talker sources these frames and the active controller is the destination. These messages do not travel all the way around the loop.

ETE - End of transmission, error. Addressed ready group, ready class, 101 01000001. This message is source by the active talker to indicate to the active controller that its device dependent transmission is finished and that one or more of the data bytes returned to the talker differently than it was sent. This error checking capability is not required, but is strongly recommended. If the device does not error check, it may not source this message. The controller replaces this message with its next operation on the loop, possibly a retry of the transmission.

ETO - End of transmission, OK. Addressed ready group, ready class, 101 01000000. This message is sourced by the active talker to indicate to the active controller that its device dependent transmission is finished and that all bytes returned to the talker as sent (or that no error checking was performed, if the device does not have the capability). Error checking is strongly recommended. The controller replaces this message with its next operation on the loop.

*GET - Group execute trigger. Addressed command group, command class, 100 00001000. GET is a command used by the controller to cause all devices which are listener addressed to begin their particular device operation. The particular operation for each device is designer specified. This permits the controller to start an operation in several devices as nearly at the same time as is possible given the loop architecture of the system.

*GTL - Go to local. Addressed command group, command class, 100 00000001. The controller uses this command to put all devices which are listener addressed under control of their front panel "local" controls. Programming data for the device will, in general, be ignored while the device is in this state, if it is received via the interface.

IAA - Illegal auto address. Auto address group, ready class, 101 10011111. When the controller assigns addresses with the AAD message, there may be exactly the maximum or too many devices on the loop. If this is the case, the message which returns to the controller has its address bit incremented to the value 31, which is defined as the

IAA message. Devices will not respond to this message. If IAA returns to the controller, it should send AADU again. If this message returns unchanged, there are exactly the maximum number of devices. The controller can start normal operations. If the frame returns incremented, there are too many devices. The controller should indicate this to the user and should not attempt any operation until the condition is corrected.

*IDY - Identify. 11X XXXXXXXX. This major classification of messages is used by the controller to poll the loop for service. Bit C1 is set by devices which need service. Additionally, if the controller has configured devices to respond to parallel poll, these devices set designated data bits in the IDY message as it passes through the device. With this capability, the controller can rapidly identify which device needs service. In general, the controller sources the IDY, but can enable certain other devices to do so if necessary.

IEP - Illegal extended primary. Auto address group, ready class, 101 10111111. This message is this same as the AEP message except that the address value is 31, an illegal value. Devices will not respond to this message. Furthermore, since AEP is never incremented, devices will not generate this message, either. It is included merely for consistency.

IES - Illegal extended secondary. Auto address group, ready class, 101 11011111. Extended address devices receive their secondary address assignments via the AES message, which they accept, increment, and send on to the next device. When the address value reaches 31, it is defined as the IES message and is no longer accepted by other devices, which merely pass it on back to the controller. The controller then assigns primary addresses with AEP message, which is not incremented. Only devices which just received the AES message will respond to the AEP message. After this group of devices has both secondary and primary addresses, it will not repond any more to the AES, so the controller sends it out again to configure the next group of devices, and so on. Should the controller run out of AEP addresses and still have the AES message return incremented, there are too many devices on the loop.

*IFC - Interface clear. Universal command group, command class, 100 10010000. IFC is sourced only by the system controller. It can be sent at any time to take over the control of the interface system. IFC resets all active devices to their idle condition, but does not destroy parallel poll or address configuration. If the active controller which is not the system controller receives the IFC, it also goes to the idle state.

IMP - Illegal multiple primary. Auto address group, ready class, 101 11111111. The controller uses the AMP message to assign primary addresses to those devices which have

multiple address capability. If there are exactly the maximum number or too many devices of this type on the loop, the AMP message will be incremented to 31, which is defined as the IMP message. No further devices will respond and the frame will return to the controller. The controller could send AMP again. If it returns unchanged, the loop has exactly the maximum number of devices. If it returns modified, there are too many. The controller should signal an error to the user and should not attempt to operate the loop.

*LADn - Listen address n command. Listen address group, command class, 100 001AAAAA. This is the command which the controller uses to cause a particular device to become the active listener, that is, able to receive data messages from other devices on the loop. AAAAA represents a five bit binary address n, which can range from 0 to 30. 31 is an illegal address which causes all listeners to go to the idle state. It is called the unlisten command. For devices which use a two byte address, LAD provides the primary address only. The MSA command must be received to make these devices an active listener. Multiple LAD messages will activate multiple listeners.

LAG - Listen address group. Command class, 100 001XXXXX. This group of commands includes all the listen address commands and also the unlisten command (address 31). Secondary address commands are in a separate group; only primary listen addresses are included in LAG.

LLO - Local lockout. Universal command group, command class, 100 00010001. With this command the controller can cause all devices which respond to this command to lock out, or not respond to, their return to local control buttons on the instruments. This will prevent an operator from changing an instrument's control settings inadvertantly at a critical time.

*LPD - Loop power down. Universal command group, command class, 100 10011011. The controller uses this command to place the loop, or rather all devices which respond to this command, in a power down state to conserve power. The controller remains powered up so that it can wake up the loop at a later time and continue normal operations. If the controller has enabled the asynchronous request mode prior to sending the LPD, other devices with the proper capability can also initiate the loop wake-up sequence.

MLA - My listen address. Listen address group, command class, 100 001MMMMM. This is the particular LADn command which happens to match in the least significant five bits with the address (primary address in the case of devices that have a two frame address) which is assigned to this specific device. This command causes the device to become the active listener and able to receive device dependent messages. In the case of devices which require two frame addresses, the MSA code is also required before the device becomes active.

MSA - My secondary address. Secondary address group, command class, 100 011MMMMM. This is the particular SADn command which matches the secondary address assigned to this specific device. This command causes the device to become the active talker or to become an active listener. This command must immediately follow the MTA or MLA command.

MTA - My talk address. Talk address group, command class, 100 010MMMMM. This is the particular TADn command which matches the talk address (primary address in the case of devices that have a two frame address) which is assigned to this specific device. When the device receives MTA, it becomes the active talker on the loop, and, when enabled, will send device dependent data to other devices. For devices which have a two byte address, the MSA command is also required before the device becomes the active talker. This command causes any previous talker to become unaddressed.

NAA - Next auto address. Auto address group, ready class, 101 100NNNNN. This mnemonic represents the incremented auto address frame which a device sends on to he next device on the loop. The value of NNNNN might range from 1 to 31, depending on the particular device and how many devices are on the loop. The value 31 is also called IAA, values less than 31 are also called AAD.

NES - Next extended secondary. Auto address group, ready class, 101 110NNNNN. This mnemonic represents the incremented secondary address assignment frame which a device passes on to the next device on the loop. It is used by extended address devices and by multiple address devices. NNNNN can range from 1 to 31, depending on the number of devices and the location on the loop. The value of 31 is also referred to as IES, values less than 31 are also called AES.

NMP - Next multiple primary. Auto address group, ready class, 101 111NNNNN. This mnemonic represents the incremented primary address frame which multiple address devices send on to the next device on the loop. NNNNN can range from 1 to 31 depending on the number of devices and the location of this device on the loop. The address value 31 is also referred to IMP and values less than 31 are also referred to as AMP.

*NRD - Not ready for data. Addressed ready group, ready class, 101 01000010. When the controller needs to synchronously interrupt a talker-listener transmission, it does so by holding the next data byte and replacing it with the NRD message. This frame signals the talker that it should terminate its transmission. When the NRD returns to the controller, it will then send the held up data frame. When this data frame returns to the talker, the talker must source the proper end of transmission frame. When the controller enables the transmission to continue, it will normally begin at the point of interruption.

NRE - Not remote enable. Universal command group, command class, 100 10010011. With this command, the controller causes all instruments to be placed under local control, that is, they will respond to their front panel controls and not to programming information received over the loop. Devices which do not implement the remote local interface function will simply ignore the command.

NUL - Null command. Addressed command group, command class, 100 00000000. This is the "no operation" command in PIL. Devices do not generally respond to this command and so it is useful for such functions as waking up a powered down loop, terminating the asynchronous request mode, etc. where it may have to be sent more than once and the devices need to remain essentially undisturbed.

OSA - Other secondary address. Secondary address group, command class, 100 011TTTTT. This mnemonic represents any secondary address command whose address bits do not match the address assigned to this particular device. It is recognized by extended talker functions to cause them to become unaddressed when another device with the same primary address is addressed to talk. TTTTT represents the address bits which do not match.

OTA - Other talk address. Talk address group, command class, 100 010TTTTT. This mnemonic represents any talk address command wherein the five address bits, TTTTT, do not match the address assigned to this particular device. Since there can only be one active talker on the loop at a time, this device will become unaddressed to talk if it was not already in the talker idle state.

PPD - Parallel poll disable. Addressed command group, command class, 100 00000101. This command is used by the controller to cause the devices which are listen addressed to no longer respond to parallel polls. The parallel poll function returns to its idle state.

PPEn - Parallel poll enable n. Addressed command group, command class, 100 1000SBBB. This command allows the controller to configure devices on the loop to respond to parallel polls in various ways. The S bit indicates the sense of the device's response. A 1 in S will cause the device to set the proper bit if it needs service, a 0 in S will cause the device to set the proper bit if it does not need service. BBB indicates the particular bit on which the device must respond. 000 indicates bit D1, 001 indicates D2, ... , 111 indicates D8. Only listen addressed devices will be configured by the PPE command. Typically, the controller will address each device individually and send the PPE command to configure it until all devices are configured.

PPU - Parallel poll unconfigure. Universal command group, command class, 100 00010101. The controller uses this command to disable all devices on the loop, whether addressed or not, from responding to parallel polls. The

parallel poll function in each device will return to its idle state.

RDY - Ready. 101 XXXXXXXX. This major class of messages is used for a number of different purposes, including device handshake functions and address configuration. Most are sourced by the active controller, but a few are sourced by talkers, and a number are modified by auto address devices.

*REN - Remote enable. Universal command group, command class, 100 10010010. With this command, the controller places the entire loop in remote mode. Now, as each device is listen addressed, it will no longer respond to its front panel controls, but will respond to programming data over the interface. Devices which do not implement this function always respond to the interface. They will simply retransmit this command and take no other action.

RFC - Read for command. Ready class, 101 00000000. The controller uses this ready frame as the handshake after each command so that it knows that all devices on the loop have received the command and are ready to receive the next. Every command must be immediately followed by the RFC message.

SADn - Secondary address n command. Secondary address group, command class, 100 011AAAAA. This command provides the secondary address to enable talkers and listeners which respond to two byte addresses. AAAAA represents the five bit binary address n, which can range from 0 to 30. 31 is an illegal address. The secondary address must follow the primary address command immediately to enable the device.

SAG - Secondary address group. Command class, 100 011XXXXX. This group of commands contains the secondary addresses, that is, SAD commands. These are only used for devices which respond to extended or multiple address modes.

SAI - Send accessory ID. Addressed ready group, ready class, 101 01100011. This frame is used by the controller to cause the addressed talker to begin sending its accessory ID byte(s). The talker replaces the SAI with the accessory ID on the loop, and terminates with the proper EOT, just as in a data transmission. If the device does not have accessory ID capability, it merely sends the SAI back to the controller, which realizes that the device cannot respond and resumes control. Accessory ID usually consists of a single byte whose high order four bits indicate the device class, for example, printer, mass storage, etc. and whose low order four bits indicate the specific device.

*SDA - Send data. Addressed ready group, ready class, 101 01100000. This frame is used by the controller to cause the addressed talker to begin sending its data string. The talker replaces the SDA with the first byte of its data on the loop and continues to send until

finished. Then it sends the proper EOT message to signal the controller that it is finished. If the device cannot source data, it merely returns the SDA to the controller.

*SDC - Selected device clear. Addressed command group, command class, 100 00000100. This command is used by the controller to clear only those devices which are listener addressed. The clear function is device dependent and has no affect on the interface functions.

SDI - Send device ID. Addressed ready group, ready class, 101 01100010. SDI is used by the controller to cause the talker addressed device to begin sending its device ID string. The talker replaces the SDI with its ID string on the loop and terminates the transmission with the proper EOT message. The ID string usually consists of ASCII characters, two for the manufacturer code, up to five for the model number, one for a model number revision letter, and any additional options or device dependent items which the designer feels should be included to clarify the identification and capabilities of this device. A CR, LF pair is usually sent just prior to the EOT. If a device does not implement device ID, the SDI will simply return to the controller.

SOT - Start of transmission. Addressed ready group, ready class, 101 01100XXX. This subgroup of frames includes the SDA, SST, SDI, SAI, and TCT messages. They all serve to enable the beginning of transmission from a device other than the controller (except for TCT, which enables the new controller). Usually, the other device will source the proper EOT message when finished, signalling the controller to take over once again. In the case of TCT (take control) the new controller remains in control of the loop and does not source the EOT. This group of frames does not go completely around the loop, but is replaced by the first frame from the enabled device.

SRQ - Service request. Data or end class or identify class, 0X1 XXXXXXXX or 111 XXXXXXXX. The SRQ bit is bit C1 of data, end, and identify messages. Devices may set this bit when they have a need for service from the controller. The bit then represents the logical OR of the various devices' need for service bits. The controller will generally need to do a serial poll operation to find out which device needs service. The command and ready classes of frames do not have a service request bit and do not transmit this message.

SST - Send status. Addressed ready group, ready class, 101 01100001. The controller uses this frame to cause the addressed talker to begin sending its status byte(s). The talker replaces the SST with its status information on the loop. When finished, the talker then transmits the proper EOT message. Bit D7 of the status byte is the bit which indicates that this device did or did not request service. Other bits may be device dependent or they may represent a coded system status message, depending on the

particular device and bit D8. If the device does not implement serial poll (status), the SST frame simply returns to the controller.

*TADn - Talk address n command. Talk address group, command class, 100 010AAAAA. The controller uses this command to enable one device on the loop to be the active talker. AAAAA represents the device's address (primary address in the case of those devices which respond to a two byte address). Address values can range from 0 to 30. 31 is an illegal address which causes any talker addressed device to return to its idle state. This code is called UNT, untalk. Since only one device can be the active talker at a time, if a device receives any address other than its own, it also becomes unaddressed.

TAG - Talk address group. Command class, 100 010XXXXX. This group includes all the talk address commands and the untalk command. Only primary talk addresses are contained in this group. Secondary addresses are contained in the SAG group.

*TCT - Take control. Addressed ready group, ready class, 101 0110010 The active controller uses this frame to pass control of the loop to another controller. The device to which control is passed must first be talk addressed, then the current controller sends the TCT message. Upon receipt fo the TCT, the talker addressed device becomes active, and replaces the TCT with its first operation on the loop as the new controller. If the device cannot accept control of the loop, it merely retranmits the TCT which returns to the current controller, which resumes active control of the loop.

UCG - Universal command group. Command class, 100 X001XXXX. This group of commands includes all those to which all devices respond whether they are currently addressed or not.

*UNL - Unlisten. Listen address group, command class, 100 00111111. This command causes all addressed listeners to return to the idle state. The controller would likely use this command to reset the listeners before addressing a new listener(s) for the next data transmission.

*UNT - Untalk. Talk address group, command class, 100 01011111. This command causes the addressed talker to return to the idle state. Since a new talk address causes the previous talker to become unaddressed anyway, this command is only useful in certain special cases, such as when it is necessary to have no talker addressed device on the loop.

ZES - Zero extended secondary. Auto address group, ready class, 101 11000000. This frame is used by the controller to assign secondary addresses to those devices which have multiple address capability. After each device has received its primary address via the AMP frame, it waits to recognize the ZES frame. When it is received, the low

C. CODING CHARTS AND FRAME HIERARCHY

DATA OR END CLASS (0XX) - ASCII

DDDD 7654 \ DDD 321	000	001	010	011	100	101	110	111
0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
0001	BS	HT	LF	VT	FF	CR	SO	SI
0010	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
0011	CAN	EM	SUB	ESC	FS	GS	RS	US
0100	SP	!	"	#	$	%	&	'
0101	(	)	*	+	,	-	.	/
0110	0	1	2	3	4	5	6	7
0111	8	9	:	;	<	=	>	?
1000	@	A	B	C	D	E	F	G
1001	H	I	J	K	L	M	N	O
1010	P	Q	R	S	T	U	V	W
1011	X	Y	Z	[	\	]	^	_
1100	`	a	b	c	d	e	f	g
1101	h	i	j	k	l	m	n	o
1110	p	q	r	s	t	u	v	w
1111	x	y	z	{	|	}	~	DEL

Bit D8 is device dependent; ASCII is a seven bit code.

CODING CHARTS AND FRAME HIERARCHY (continued)

DDDDD 87654	DDD 321 000	001	010	011	100	101	110	111	GROUP
	COMMAND CLASS (100)								
00000	NUL	GTL	.	.	SDC	PPD	.	.	ACG
00001	GET	.	.	.	.	.	.	.	ACG
00010	.	LLO	.	.	DCL	PPU	.	.	UCG
00011	EAR	.	.	.	.	.	.	.	UCG
00100	LAD0	LAD1	LAD2	LAD3	LAD4	LAD5	LAD6	LAD7	LAG
00101	LAD8	LAD9	LAD10	LAD11	LAD12	LAD13	LAD14	LAD15	LAG
00110	LAD16	LAD17	LAD18	LAD19	LAD20	LAD21	LAD22	LAD23	LAG
00111	LAD24	LAD25	LAD26	LAD27	LAD28	LAD29	LAD30	UNL	LAG
01000	TAD0	TAD1	TAD2	TAD3	TAD4	TAD5	TAD6	TAD7	TAG
01001	TAD8	TAD9	TAD10	TAD11	TAD12	TAD13	TAD14	TAD15	TAG
01010	TAD16	TAD17	TAD18	TAD19	TAD20	TAD21	TAD22	TAD23	TAG
01011	TAD24	TAD25	TAD26	TAD27	TAD28	TAD29	TAD30	UNT	TAG
01100	SAD0	SAD1	SAD2	SAD3	SAD4	SAD5	SAD6	SAD7	SAG
01101	SAD8	SAD9	SAD10	SAD11	SAD12	SAD13	SAD14	SAD15	SAG
01110	SAD16	SAD17	SAD18	SAD19	SAD20	SAD21	SAD22	SAD23	SAG
01111	SAD24	SAD25	SAD26	SAD27	SAD28	SAD29	SAD30	.	SAG
10000	PPE01	PPE02	PPE03	PPE04	PPE05	PPE06	PPE07	PPE08	ACG
10001	PPE11	PPE12	PPE13	PPE14	PPE15	PPE16	PPE17	PPE18	ACG
10010	IFC	.	REN	NRE	.	.	.	.	UCG
10011	.	.	AAU	LPD	.	.	.	.	UCG
10100	DDL0	DDL1	DDL2	DDL3	DDL4	DLL5	DLL6	DLL7	ACG
10101	DDL8	DDL9	DDL10	DDL11	DDL12	DDL13	DDL14	DDL15	ACG
10110	DDL16	DDL17	DDL18	DDL19	DDL20	DDL21	DDL22	DDL23	ACG
10111	DDL24	DDL25	DDL26	DDL27	DDL28	DDL29	DDL30	DDL31	ACG
11000	DDT0	DDT1	DDT2	DDT3	DDT4	DDT5	DDT6	DDT7	ACG
11001	DDT8	DDT9	DDT10	DDT11	DDT12	DDT13	DDT14	DDT15	ACG
11010	DDT16	DDT17	DDT18	DDT19	DDT20	DDT21	DDT22	DDT23	ACG
11011	DDT24	DDT25	DDT26	DDT27	DDT28	DDT29	DDT30	DDT31	ACG
11100	.	.	.	.	.	.	.	.	
11101	.	.	.	.	.	.	.	.	
11110	.	.	.	.	.	.	.	.	
11111	.	.	.	.	.	.	.	.	

CODING CHARTS AND FRAME HIERARCHY (continued)

READY CLASS (101)

DDDDD 87654 \ DDD 321	000	001	010	011	100	101	110	111	GROUP
00000	RFC	.	.	.	.	.	.	.	RFC
00001	.	.	.	.	.	.	.	.	RFC
00010	.	.	.	.	.	.	.	.	RFC
00011	.	.	.	.	.	.	.	.	RFC
00100	.	.	.	.	.	.	.	.	
00101	.	.	.	.	.	.	.	.	
00110	.	.	.	.	.	.	.	.	
00111	.	.	.	.	.	.	.	.	
01000	ETO	ETE	NRD	.	.	.	.	.	ARG
01001	.	.	.	.	.	.	.	.	ARG
01010	.	.	.	.	.	.	.	.	ARG
01011	.	.	.	.	.	.	.	.	ARG
01100	SDA	SST	SDI	SAI	TCT	.	.	.	ARG
01101	.	.	.	.	.	.	.	.	ARG
01110	.	.	.	.	.	.	.	.	ARG
01111	.	.	.	.	.	.	.	.	ARG
10000	AAD0	AAD1	AAD2	AAD3	AAD4	AAD5	AAD6	AAD7	AAG
10001	AAD8	AAD9	AAD10	AAD11	AAD12	AAD13	AAD14	AAD15	AAG
10010	AAD16	AAD17	AAD18	AAD19	AAD20	AAD21	AAD22	AAD23	AAG
10011	AAD24	AAD25	AAD26	AAD27	AAD28	AAD29	AAD30	IAA	AAG
10100	AEP0	AEP1	AEP2	AEP3	AEP4	AEP5	AEP6	AEP7	AAG
10101	AEP8	AEP9	AEP10	AEP11	AEP12	AEP13	AEP14	AEP15	AAG
10110	AEP16	AEP17	AEP18	AEP19	AEP20	AEP21	AEP22	AEP23	AAG
10111	AEP24	AEP25	AEP26	AEP27	AEP28	AEP29	AEP30	IEP	AAG
11000	AES0	AES1	AES2	AES3	AES4	AES5	AES6	AES7	AAG
11001	AES8	AES9	AES10	AES11	AES12	AES13	AES14	AES15	AAG
11010	AES16	AES17	AES18	AES19	AES20	AES21	AES22	AES23	AAG
11011	AES24	AES25	AES26	AES27	AES28	AES29	AES30	IES	AAG
11100	AMP0	AMP1	AMP2	AMP3	AMP4	AMP5	AMP6	AMP7	AAG
11101	AMP8	AMP9	AMP10	AMP11	AMP12	AMP13	AMP14	AMP15	AAG
11110	AMP16	AMP17	AMP18	AMP19	AMP20	AMP21	AMP22	AMP23	AAG
11111	AMP24	AMP25	AMP26	AMP27	AMP28	AMP29	AMP30	IMP	AAG

CODING CHARTS AND FRAME HIERARCHY (continued)

```
DOE .... DAB
       . DAB(SRQ)
       . END
       . END(SRQ)

CMD .... ACG .... NUL
       .        . GTL
       .        . SDC
       .        . PPD
       .        . GET
       .        . PPE(0-15)
       .        . DDL(0-31)
       .        . DDT(0-31)
       .
       . UCG .... LLO
       .        . DCL
       .        . PPU
       .        . EAR
       .        . IFC
       .        . REN
       .        . NRE
       .        . AAU
       .        . LPD
       .
       . LAG .... LAD(0-30)
       .        . MLA(0-30)
       .        . UNL(31)
       .
       . TAG .... TAD(0-30)
       .        . MTA(0-30)
       .        . OTA(0-30)
       .        . UNT(31)
       .
       . SAG .... SAD(0-30)
                . MSA(0-30)
                . OSA(0-30)

IDY .... IDY
       . IDY(SRQ)

RDY .... RFC
       .
       . ARG .... EOT .... ETO
       .                 . ETE
       .        .
       .        . NRD
       .        .
       .        . SOT .... SDA
       .                 . SST
       .                 . SDI
       .                 . SAI
       .                 . TCT
       .
       . AAG .... AAD(0-30)
                . NAA(1-31)
                . IAA(31)
                .
                . AEP(0-30)
                . IEP(31)
                .
                . ZES(0)
                . AES(0-30)
                . NES(1-31)
                . IES(31)
                .
                . AMP(0-30)
                . NMP(1-31)
                . IMP(31)
```

D. ACCESSORY ID AND STATUS MESSAGE DEFINITIONS

The accessory ID provides a fast, efficient means for simple controllers to identify specific devices and device classes. The high order four bits indicate the general device class while the low order four bits indicate the specific device. In general, only one byte is sent, but additional bytes may be necessary under certain circumstances. The code 1111 in both the class and specific device fields is reserved as an extend code which is presently not defined. Presently defined classes and devices are as follows:

class		device
0000	controllers	0000 HP 41C no others presently defined
0001	mass storage devices	0000 HP 82161A minicassette tape drive no others presently defined
0010	printers	0000 HP 82162A 24 column thermal printer
0011	displays	none presently defined
0100	interfaces	0000 HP 82165A/82166A general purpose interface no others presently defined
0101	instruments	none presently defined
0110	graphic i/o	none presently defined

no others presently defined

The status byte(s) provides the controller with the ability to determine which devices require service and to effectively control the other device functions. Bit D7 of the first status byte returned to the controller is always reserved to indicate that this device needs service (1) or that it does not (0). In general, the other bits of the first byte and all bits of any other bytes are designer specified. It is recommended that devices follow this convention, however: If bit D8 is a zero, all other bits (except D7, of course) are designer specified. If

bit D8 is a 1, this is a flag to indicate that the least significant six bits in the first byte are a binary coded, system defined status message number. Any succeeding bytes are designer specified. If a device responds to accessory ID, however, it must adhere to this convention. The presently defined system status messages are as follows:

DDDDDD 654321	message
000000	all OK, this device does not require service (bit D7 will also be zero)
000001	low battery
000010	require manual intervention
100000	request control of the loop
111111	ASCII message follows (before EOT)